THE UNIVERSITY *of* EDINBURGH

High Performance Reconfigurable Architectures for Biological Sequence Alignment

Mohd Nazrin Md Isa

A thesis submitted for the Degree of Doctor Philosophy

The University Of Edinburgh

March 2013

Declaration of Originality

I hereby declare that, this thesis and the research work reported herein was composed and originated entirely by myself in the School of Engineering at the University of Edinburgh.

Mohd Nazrin Md Isa
March 2013
Edinburgh, Scotland
United Kingdom

This thesis is dedicated to:

My dear parents

My lovely wife

My lovely son, daughter and the little twins

My dear brothers and sisters

My lovely nephews and nieces

Thanks a lot for the endless support, love and encouragements!

Acknowledgements

All praises due to Allah, the most gracious, the most merciful. I would like to take this opportunity to express my sincere thanks to those who have involved and supported me towards the successful completion of my PhD study.

First and foremost, my deepest gratitude goes to my supervisor, Dr. Khaled Benkrid. Thank you very much indeed, for his suggestions, criticisms and guidance. His invaluable efforts, profound expertise, patience and kindness towards developing my professional knowledge especially in the field of reconfigurable computing, hardware designs and technical writing skills are highly appreciated.

Secondly, this appreciation goes to Dr. Thomas Clayton, who has introduced me fundamental aspects of hardware description language and hardware debugging skills. In addition, this appreciation also goes to Dr. Ahmet T. Erdogan for his valuable input and kindness. To all my friends in the System Level Integration Group (SLig), your helps and encouragements are greatly appreciated.

Special thanks also to my financial sponsor, University Malaysia Perlis (UniMAP) and Ministry of Higher Education Malaysia (MoHE) for the scholarship and administrative supports. Also thanks to the University of Edinburgh for the facilities and administrative supports during my study.

Last but not least, this special appreciation also goes to my lovely family and my dear parents for their love, encouragements and endless support throughout the years of effort.

Thank you – Terima kasih – Gracias –Merci–-Jazakallah

Abstract

Bioinformatics and computational biology (BCB) is a rapidly developing multidisciplinary field which encompasses a wide range of domains, including genomic sequence alignments. It is a fundamental tool in molecular biology in searching for homology between sequences. Sequence alignments are currently gaining close attention due to their great impact on the quality aspects of life such as facilitating early disease diagnosis, identifying the characteristics of a newly discovered sequence, and drug engineering. With the vast growth of genomic data, searching for a sequence homology over huge databases (often measured in gigabytes) is unable to produce results within a realistic time, hence the need for acceleration. Since the exponential increase of biological databases as a result of the human genome project (HGP), supercomputers and other parallel architectures such as the special purpose Very Large Scale Integration (VLSI) chip, Graphic Processing Unit (GPUs) and Field Programmable Gate Arrays (FPGAs) have become popular acceleration platforms. Nevertheless, there are always trade-off between area, speed, power, cost, development time and reusability when selecting an acceleration platform. FPGAs generally offer more flexibility, higher performance and lower overheads. However, they suffer from a relatively low level programming model as compared with off-the-shelf microprocessors such as standard microprocessors and GPUs. Due to the aforementioned limitations, the need has arisen for optimized FPGA core implementations which are crucial for this technology to become viable in high performance computing (HPC).

This research proposes the use of state-of-the-art reprogrammable system-on-chip technology on FPGAs to accelerate three widely-used sequence alignment algorithms; the Smith-Waterman with affine gap penalty algorithm, the profile hidden Markov model (HMM) algorithm and the Basic Local Alignment Search Tool (BLAST) algorithm. The three novel aspects of this research are firstly that the algorithms are designed and implemented in hardware, with each core achieving the highest performance compared to the state-of-the-art. Secondly, an efficient scheduling strategy based on the double buffering technique is adopted into the hardware architectures. Here, when the alignment matrix computation task is overlapped with the PE configuration in a folded systolic array, the overall throughput of the core is significantly increased. This is due to the bound PE configuration time and the parallel PE configuration approach irrespective of the number of PEs in a systolic array. In addition, the use of only two configuration elements in the PE

optimizes hardware resources and enables the scalability of PE systolic arrays without relying on restricted onboard memory resources. Finally, a new performance metric is devised, which facilitates the effective comparison of design performance between different FPGA devices and families. The normalized performance indicator (speed-up per area per process technology) takes out advantages of the area and lithography technology of any FPGA resulting in fairer comparisons.

The cores have been designed using Verilog HDL and prototyped on the Alpha Data ADM-XRC-5LX card with the Virtex-5 XC5VLX110-3FF1153 FPGA. The implementation results show that the proposed architectures achieved giga cell updates per second (GCUPS) performances of 26.8, 29.5 and 24.2 respectively for the acceleration of the Smith-Waterman with affine gap penalty algorithm, the profile HMM algorithm and the BLAST algorithm. In terms of speed-up improvements, comparisons were made on performance of the designed cores against their corresponding software and the reported FPGA implementations. In the case of comparison with equivalent software execution, acceleration of the optimal alignment algorithm in hardware yielded an average speed-up of 269x as compared to the SSEARCH 35 software. For the profile HMM-based sequence alignment, the designed core achieved speed-up of 103x and 8.3x against the HMMER 2.0 and the latest version of HMMER (version 3.0) respectively. On the other hand, the implementation of the gapped BLAST with the two-hit method in hardware achieved a greater than tenfold speed-up compared to the latest NCBI BLAST software. In terms of comparison against other reported FPGA implementations, the proposed normalized performance indicator was used to evaluate the designed architectures fairly. The results showed that the first architecture achieved more than 50 percent improvement, while acceleration of the profile HMM sequence alignment in hardware gained a normalized speed-up of 1.34. In the case of the gapped BLAST with the two-hit method, the designed core achieved 11x speed-up after taking out advantages of the Virtex-5 FPGA. In addition, further analysis was conducted in terms of cost and power performances; it was noted that, the core achieved 0.46 MCUPS per dollar spent and 958.1 MCUPS per watt. This shows that FPGAs can be an attractive platform for high performance computation with advantages of smaller area footprint as well as represent economic ‘green’ solution compared to the other acceleration platforms. Higher throughput can be achieved by redeploying the cores on newer, bigger and faster FPGAs with minimal design effort.

Table of Contents

List of Figures

List of Tables

Acronyms and Abbreviations

ASIC	Application Specific Integrated Circuit
ASIP	Application Specific Integrated Processor
BCB	Bioinformatics and Computational Biology
BioSCAN	Biological Sequence Comparative Analysis Node
BISP	Biological Information Signal Processor
BLAST	Basic Local Search Tool
BLOSUM	BLOcks SUbstitution Matrices
CAD	Computer Aided Design
CE	Configuration Element
CLB	Configurable Logic Block
CPLD	Complex Programmable Logic Devices
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
CUPS	Cell Update Per Second
DMA	Direct Memory Access
DNA	Deoxyribonucleic acid
DP	Dynamic Programming
DPR	Dynamic Partial Reconfiguration
DSP	Digital Signal Processing
EMBL	European Molecular Biology Laboratory
ESL	Electronic System Level
FASTA	Fast Alignment
FF	Flip-Flop
FPGA	Field Programmable Gate Array
GB	Giga Byte
GPP	General Purpose Processor
GPU	Graphic Processor Unit
HDL	Hardware Description Language
HGP	Human Genome Project
HLL	High Level Language
HMM	Hidden Markov Model
HPC	High Performance Computing
HSP	High Scoring Pair
ICAP	Internal Configuration Access Port
IP	Intellectual Property
JTAG	Joint Test Action Group

LAB	Logic Array Block
LC	Logic Cell
LE	Logic Element
LUT	Look-up Table
MAC	Multiply-and-Accumulate Unit
NCD	Native Circuit Description
NGD	Native Generic Database
NIH	National Institute of health
NW	Needleman-Wunsch
OCC	Overlapped Computation and Configuration
PAM	Percent Accepted Mutation
PAR	Place and Route
PCAP	Parallel Configuration Access Port
PCI	Peripheral Controller Interface
PE	Processing Element
PLD	Programmable Logic Devices
PMC	PCI Mezzanine Card
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
RNA	Ribonucleic Acid
ROM	Read Only Memory
RTL	Register Transfer Logic
RTR	Run Time Reconfiguration
SAMBA	Systolic Accelerator for Molecular Biological Applications
SIMD	Single Instruction Multiple Data
SOPC	System on Programmable Chip
SRAM	Static Random Access Memory
SW	Smith-Waterman
UCF	User Constrain File
UniProtKB	Universal Protein Knowledgebase
XPS	Xilinx Platform Studio

List of Symbols

CB_x	circular buffer number *x*, *x = 0, 1, 2 …*
cdw	compute data width
CE_{Depth}	depth of a configuration element (CE)
CE_x	configuration element number *x*, *x = 0, 1, 2 …*
Cfg	configuration
d	gap-open penalty
D_{length}	length of subject sequence length in residues
D(i,j)	cost of residue deletion up to prefixes i^{th} and j^{th}
dw	data width
e	gap-extend penalty
E(i,j)	the best score up to prefixes *i* and *j*
$e(I_j,s[i])$	insertion's emission probability score of i^{th} residue in sequence *s*
$e(M_j,s[i])$	match's emission probability score of i^{th} residue in sequence *s*
F(i,j)	alignment matrix score up to prefixes i^{th} and j^{th}
F_x	fold computation number *x*
g	length of gap in sequence alignment
gap	number of gaps in alignment
gdw	gap data width
I(i,j)	cost of residue insertion up to prefixes i^{th} and j^{th}
Ix(I,j)	insertion score (*x*-direction)
Iy(I,j)	insertion score (*y*-score)
k	number of folds
LC_{ratio}	ratio of logic cells
L_m	length of a profile HMM
L_s	length of subject sequence in residues
$LUT\ Delay_{\ ratio}$	look-up table delay
m	match score
M(i,j)	alignment matrix score up to prefixes i^{th} and j^{th}
match	number of matched residues under comparison
mismatch	number of mismatched residues under comparison
n_{CB}	number of circular buffers in a substitution matrix loader
n_{col}	number of columns in a substitution matrix

n_{CE}	number of configuration elements (CEs)
n_{PE}	number of processing elements (PEs)
n_{row}	number of rows
PE_0	processing element number x, $x = 0, 1, 2 \ldots$
penalty (g)	total gap penalty
Q[i]	query residue i^{th}
Q_{length}	length of a query sequence in residues
s	mismatch score
$s(x_i,y_j)$	Substitution matrix score of residue x_i and y_j
S[j]	subject sequence residue j^{th}
score	alignment score
S_{HF}	hit finder score
speed up	raw speed up calculated based on the ratio of execution times
$speed\ up_{FPGAorGPU}$	speed up of FPGA or speed up of GPU
$speed\ up_{AreaNormalized}$	raw speed up normalized to the area used
$speed\ up_{Normalized}$	the area normalized speed up normalized with respect to fabrication technology
T	threshold
t_{BLASTp}	execution time for the BLASTp algorithm
tCE_{config}	time required to update configuration element (CE) with coefficients
t_{config}	initial configuration time of the emission an transition loader
t_{FPGA}	FPGA execution time
$T_{GPPorGPU}$	execution time of GPP or GPU
$t_{initialload}$	initial configuration time of the substitution matrix loader
t_{query}	time required to load query residue
$tr(B,M_j)$	probability score as a result of transition from state *B* to *M*
$tr(D_{j-1},D_j)$	probability score as a result of transition from state *D* to *D*
$tr(D_{j-1},M_j)$	probability score as a result of transition from state *D* to *M*
$tr(I_j,I_j)$	probability score as a result of transition from state *I* to *I*
$tr(I_{j-1},M_j)$	probability score as a result of transition from state *I* to *M*
$tr(M_j,I_j)$	probability score as a result of transition from state *M* to *I*
$tr(M_{j-1},D_j)$	probability score as a result of transition from state *M* to *D*
$tr(M_{j-1},E)$	probability score as a result of transition from state *M* to E
$tr(M_{j-1},M_j)$	probability score as a result of transition from state *M* to *M*
W	length of residues in the BLAST algorithm
wl	word length
x	query sequence
y	subject sequence

Chapter 1
Introduction and Motivation

Bioinformatics refers to the analysis and management of biological information, whereas computational biology is a discipline related to physical and mathematical simulations of biological processes [1]. Bioinformatics and computational biology (BCB), brings together computer scientists and molecular biologists into a single discipline which bridges the knowledge gap between hardware designs on the one hand and molecular biology on the other. The area of bioinformatics involves three important challenges [1]: Firstly, it requires a stored and organized genomic database. Secondly, it needs resources and tools to facilitate the analysis of the gathered biological data. Finally, it uses the developed tools to interpret biological information in a meaningful manner for wide ranges of important applications including in forensic, medical sciences and in facilitating the discovery of drugs. For instance, understanding the genetic and protein related information in biological sequences can lead to better medicines and treatments. Due to its potential for improving quality of life, this multidisciplinary discipline has gained in popularity over the last decade as a result of advancements in computing technologies and the successful completion in 2003 of the human genome project (HGP).

The challenge for bioinformatics has now shifted from the genomic data sets gathering in computerized databases to the techniques used to process the gathered biological information such as *deoxyribonucleic* acid (DNA) and protein sequences. Sequence alignment is one of bioinformatics disciplines, which is a fundamental tool in molecular biology used to analyze biological information. It falls under two broad classes; the optimal and heuristic-based sequence alignments. The former guaranteed the most sensitive search algorithms, however due to the intensive search approach of these algorithms, performing homology search against a huge database sequence using a standard desktop computer unable to produce results in realistic time. Alternatively, the heuristic-based approach is used to get results faster than the optimal ones; however

these types of algorithms are less sensitive. With the huge sizes of genomic databases, which are often measured in gigabytes, and exponential growth of such databases over the years, performing sequence alignment using a standard desktop computer quickly became an issue even with the heuristic approach. The vital importance of sequence alignment has led to a tremendous growth in researches, where it is crucial that biological information can be searched and analyzed using significantly more efficient and specialized tools in realistic time scale. In this research work, both of the aforementioned sequence homology search methodologies are accelerated using the field programmable gate array (FPGA) to evaluate the efficiency and advantages of the reconfigurable logic device as a viable alternative in scientific computing.

1.1 Objectives and contributions

1.1.1 Objective

Field programmable gate arrays (FPGAs) and graphic processing units (GPUs) have become two most promising accelerators for biological sequence alignments. In general, FPGAs are more flexible, and give higher performance with lower overheads, while GPUs tend to be easier to program and are less costly. Rapid advances in FPGAs have produced extremely good computation and speed performance due to the ability to exploit parallelism. However, FPGAs suffer from a relatively low level programming model as compared with off-the shelf standard and application-specific microprocessors such as GPUs. This raises the need for optimized FPGA core implementations which are crucial for this technology to become viable in high performance computing (HPC). Therefore, this thesis proposes the use of state-of-the-art reprogrammable system-on-chip technology, in the form of FPGAs, as a relatively low cost, high performance and reprogrammable implementation platform for biological sequence alignment. This research aims to develop a library of sophisticated FPGA-based biological sequence alignment core architectures of the following optimal and heuristic-based sequence alignment algorithms:

- The Smith-Waterman with affine gap penalty (optimal)
- The Profile hidden Markov model (heuristic)
- The Gapped BLAST with the two-hit method (heuristic)

1.1.2 Contributions

This research sets out to examine the advantages of FPGAs in accelerating the three widely-used sequence alignment algorithms: Dynamic programming-based sequence alignment, profile HMM-based sequence alignment and heuristics-based sequence alignment. The novel aspects of this research work are threefold:

Firstly, the novel architectures of the three sequence alignment algorithms are designed and implemented in hardware, with each achieving the highest performance against state-of-the-art. The first architecture uses the well-known Smith-Waterman algorithm with an affine gap penalty to align biological sequences with optimal results, while the second architecture uses the profile hidden Markov model of multiple sequence alignments to search for sequence homologies in a database. Finally, the heuristic-based sequence alignment focuses on the gapped BLAST with two-hit method for faster sequence homologies search.

Secondly, an efficient scheduling strategy based on the double buffering technique is adopted in the core architectures. Typically, a PE holds one character of a query sequence for alignment matrix computation. However, biological sequences are often longer than the number of processing elements (PEs) available in an FPGA. Therefore an efficient scheduling strategy is proposed for the core architectures in order to re-use PE systolic arrays for the multiple pass processing of such biological sequences without requiring additional time for PE configuration. This is accomplished by designing a parallel loader to configure PEs in bounded configuration time regardless of their number. This allows for the time-consuming task of the alignment matrix computation to be overlapped with the PE configuration of the subsequent fold computation in a folded systolic array. This way, the overlapping operation significantly increases the overall system throughput. Moreover, the use of only two configuration elements (CEs) per PE optimizes FPGA resources and enables the scalability of PE systolic arrays without relying on the restricted onboard memory resources.

Finally, a new performance metric is devised in this work, which facilitates the effective comparison of design performance across different FPGA devices and families. The primitive element in FPGA known as the logic cell and the ratio of internal look-up table delays of the respective FPGAs used for comparison are taken into consideration as normalization factors in the new performance metric. The normalized performance

indicator i.e. speed-up per area per process technology takes out advantages of the area and lithographic technology of any particular FPGA, resulting fairer comparison with other devices.

1.2 Thesis structure

This thesis is organized as follows:

Chapter 2 introduces the general fields of bioinformatics and computational biology and then focuses on biological sequence alignments. In particular, three widely-used types of sequence alignment algorithms are discussed: The dynamic programming-based sequence alignment, the profile HMM-based sequence alignment and the heuristic-based sequence alignment.

Chapter 3 gives a brief historical introduction to FPGAs, followed by an explanation of the generic architecture of modern FPGAs and an emphasis on the Xilinx FPGA architecture. Then, details of architecture, development tools and performance associated with FPGAs are presented. The Alpha Data Card with the Virtex-5 FPGA on it, which is used in this research, is then discussed.

Chapter 4 details the design and corresponding hardware implementation of the dynamic programming-based sequence alignment algorithm in hardware. Prior to that, the background of the DP-based algorithm and relevant previous work are discussed, followed by a description and elaboration of the systolic array architecture, which is widely-used to accelerate dynamic programming algorithms in hardware. Then, implementation results of the corresponding hardware architecture are compared against software and other FPGA implementations. Towards the end of this chapter, conclusions and future work for the optimal sequence alignment are laid out.

Chapter 5 focuses on the design and hardware implementation of the profile hidden Markov model-based sequence alignment. The background of the profile HMM, which models the specific positions of multiple sequence alignments is first elaborated. Then earlier work on the FPGA implementations of this type of sequence alignment is described and the corresponding hardware architecture of the profile-to-sequence alignment with speculative systolic array computation is presented. Following that, the performance of the proposed core is discussed by comparing it with the latest HMMER package and other reported FPGA implementations.

Chapter 6 presents the final architecture considered in this research. It first introduces the background of the basic local alignment search tool (BLAST). Then, prior work which focuses on other reported FPGA-based gapped BLAST using the two-hit method is discussed. Following this, the novel hardware architecture of the gapped BLAST with the two-hit method is proposed. Towards the end of this chapter, the implementation results for the designed core are presented before conclusions are drawn and suggestions for future work.

Chapter 7 evaluates the efficiency of FPGAs in terms of area, speed, power, energy and costs as compared to GPUs and GPPs particularly for biological sequence alignment. Then, based on these criteria, the normalized performance per dollar and energy spent for the respective implementation platforms are calculated. Finally, the question of whether or not FPGAs can be justified as a viable alternative for biological sequence alignment compared to GPUs and GPPs is considered followed by a summary and the conclusions of the chapter.

Chapter 8 summarizes the work involved in this research and outlines the conclusions of this study. Recommendations for potential future research directions are then laid out.

Chapter 2
Introduction to Biological Sequence Alignment

This chapter gives a background of biological sequence alignment and example of its application. Then, discussion of the well-known dynamic programming and heuristic algorithms used in sequence alignment algorithms is followed by a description of the biological databases widely-used in sequence alignments. Towards the end of this chapter, summary and conclusions are laid out.

2.1 Genomic data

Cells in living organisms i.e. humans, plants and animals are essentially made up of protein and nucleic acids (deoxyribonucleic acid (DNA) and ribonucleic acid (RNA)). DNA is molecule that provides the instructions required in the nucleus of every cell for it to perform biological operations. DNA comprises of four nucleic acids; adenine (A), guanine (G), cytosine (C), and thymine (T). Genes are made-up of DNAs and each gene provides instructions for cells in living organisms to make other molecules known as proteins. In the human body, the size of a gene varies from several hundred DNA residues to more than 2 million residues or bases. Proteins essentially comprise combinations any of the 20 main amino acids and arise from DNAs through two processes; the transcription process followed by the translation process. During the transcription stage, DNA information is transferred to a molecule called messenger ribonucleic acid (mRNA). The mRNA is a single-stranded copy of the gene and it is translated into a protein molecule through the translation process. This translation stage is the second step of gene expression, where the mRNA is read following the rules of the translation of the four-letter DNA code into the 20 main amino acids as shown in Figure 2.1. Their three-letter (genetic code), the equivalent name of amino acid and its single-letter code representation are summarized in Table 2.1.

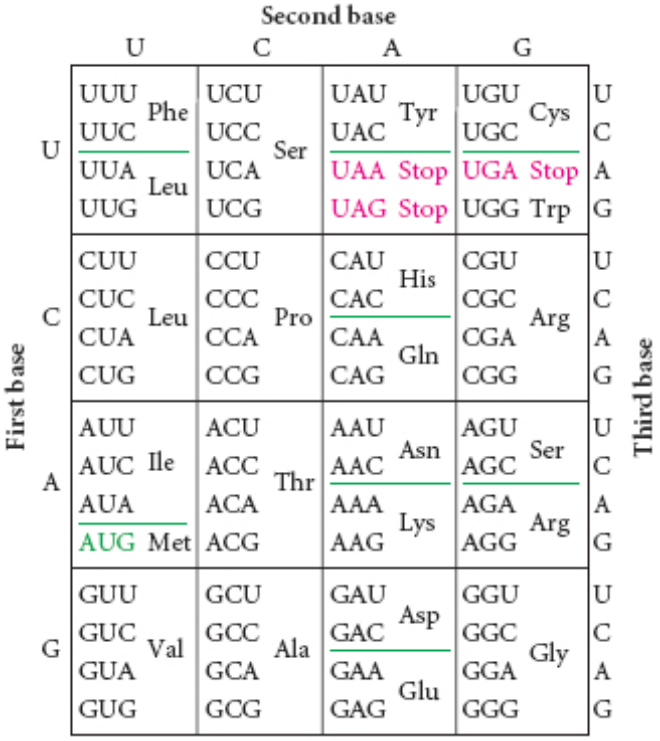


Figure 2.1: The genetic codes which relate the DNA to the amino acids [2]

Table 2.1: The main amino acids and their corresponding three-letter (genetic code) and single-letter codes [2]

Three Letter code	**Amino Acid**	**Single Letter Code**	**Three Letter code**	**Amino Acid**	**Single Letter Code**
Ala	*Alanine*	*A*	Met	*Methionine*	*M*
Cys	*Cysteine*	*C*	Asn	*Asparagine*	*N*
Asp	*Aspartic acid*	*D*	Pro	*Proline*	*P*
Glu	*Glutamic acid*	*E*	Gln	*Glutamine*	*Q*
Phe	*Phenylalanine*	*F*	Arg	*Arginine*	*R*
Gly	*Glycine*	*G*	Ser	*Serine*	*S*
His	*Histidine*	*H*	Thr	*Threonine*	*T*
Ile	*Isoleucine*	*I*	Val	*Valine*	*V*
Lys	*Lysine*	*K*	Trp	*Tryptophan*	*W*
Leu	*Leucine*	*L*	Tyr	*Tyrosine*	*Y*

Combinations of these 20 bases or residues (*A*, *C*, *D*, *E*, *F*, *G*, *H*, *I*, *K*, *L*, *M*, *N*, *P*, *Q*, *R*, *S*, *T*, *V*, *W* and *Y*), produce different protein sequences, each of which has its own biological functionality. For instance, blood in the human body contains cells known as

red blood cells that transport oxygen. The cells use a protein called hemoglobin to capture and carry oxygen around the body. In reality, biological processes such as mutation and selection cause changes in the genetic codes of DNA and proteins. Consequently these changes alter characteristics and functions of cells in living organisms. In bioinformatics, specialized tools known as sequence alignments are used to identify damages (changes in genetic codes) to biological sequences as a result of the aforementioned biological processes. The operation of sequence alignment is fundamentally to determine whether biological sequences are biologically related or have occurred by chance [3]. In pairwise sequence alignment, a newly discovered biological sequence also known as query sequence is compared against each subject sequence in a database, while for multiple sequence alignment, a query sequence is compared against many sequences at once. The rationale behind this fundamental operation is mainly to discover regions of similarity between the sequences under study, which may provide additional information on their functional, structural, evolutionary or other interesting characteristics. This is due to the fact that biological sequences have diverged from a common ancestry as a result of the aforementioned biological processes [3]. The mutational process, for instance, involves residue substitution, the insertion of new residues or the deletion of existing residues in a sequence. Substitution is the change of a residue in a sequence from one to another, while residue insertions or deletions are referred to as gaps. A gap allows an alignment between sequences to fit or conform to underlying biological models. For example, the DNA sequence ‘*C’‘A’‘G’‘T*’ could arise as a result of the residue ‘*T*’ being inserted into the sequence ‘*C’‘A’‘G*’ or the residue ‘*T*’ being deleted from the sequence ‘*C’‘A’‘G’‘T’‘T*’.

The search for sequence homology is a fundamental tool in molecular biology which makes it possible to achieve other goals such as facilitating drug engineering, the determination of a protein’s function from a sequence of amino acids, genomic sequencing, and the construction of evolutionary trees. Figure 2.2 illustrates an example of the application of sequence alignment used to facilitate drug discovery. In this example, the query sequence is a human amino acid sequence with its portion of gene has protein damage (not shown for simplicity), which occurred during DNA transcription. Transcription is the process of transcribing genetic information from DNA to produce the human protein in this example. In pairwise sequence alignment, the query sequence is searched against other sequences (HUMAN, FLY and BACTERIA) in the

database. Based the homology search, a portion of HUMAN sequence in the database is found to be identical to the query sequence as illustrated in Figure 2.2. Then, based on biological characteristics of the best matched sequence, the damaged structure of the HUMAN protein can be modeled, leading to the design of molecules as a drug which could bind the damaged HUMAN protein structure.

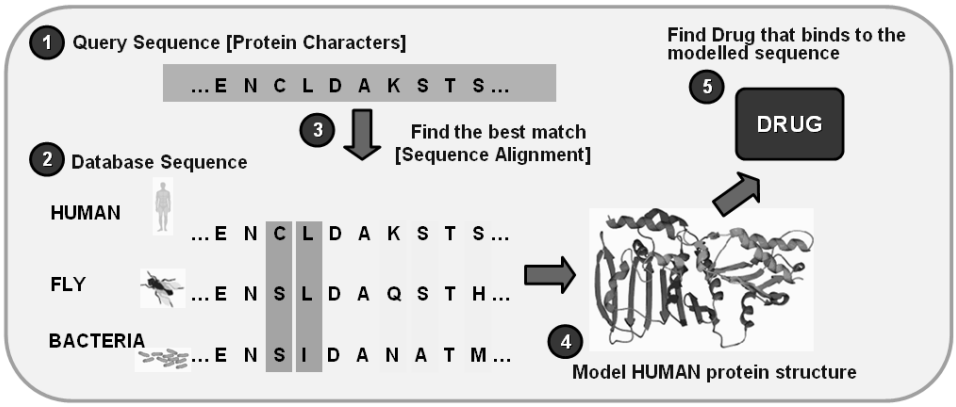


Figure 2.2: Example of the application of biological sequence alignment [4]

2.2 Genomic database

The example of pairwise sequence alignment introduced in section 2.1 only involved three subject sequences. In reality, there are often more than a million biological sequences in a database. Due to the completion of the Human Genome Project in 2003, biological information including the nucleotide and protein sequences are successfully stored, organized and indexed in computerized database with each sequence in the database is indexed with a unique identifier known as accession number. The GenBank and the Universal Protein Resource (UniProt) are examples of other well-known computerized and publicly available biological databases. The former is a comprehensive genetic sequence database developed by the United States National Center for Biotechnology Information (NCBI) at the National Institute of Health (NIH). It is an archive of primary sequence data [5] and as of April 2011, the GenBank contained nucleotide sequences of more than 380,000 organisms, with 135,440,924 sequence entries [6]. The GenBank also has two daily data exchange members under the International Nucleotide Sequence Database Collaboration (INSDC); the DNA databank

of Japan (DBBJ) and the European Molecular Biology Laboratory (EMBL). The GenBank provides both protein and DNA data, while the Uniprot provides protein data only [5].

The UniProt was initially formed through three separate protein databases providers; the Swiss Institute of Bioinformatics and the European Bioinformatics Institute (EBI), the Translated EMBL Nucleotide Sequence Data Library (TrEMBL) databases, and the Georgetown University's Protein Information Resource - Protein Sequence Database (PIR-PSD) [5]. Then the Uniprot and the TrEMBL continue as two separate entities in the UniProt knowledge base (UniprotKB). The former is manually annotated protein database and referred to as UniProtKB/Swiss-Prot while the latter is computationally analyzed sequence record from the INSDC and known as UniProtKB/TrEMBL [7]. Figure 2.3 shows the distribution of biological sequences by length (number of residues) in the UniProtKB/Swiss-Prot database as an example.

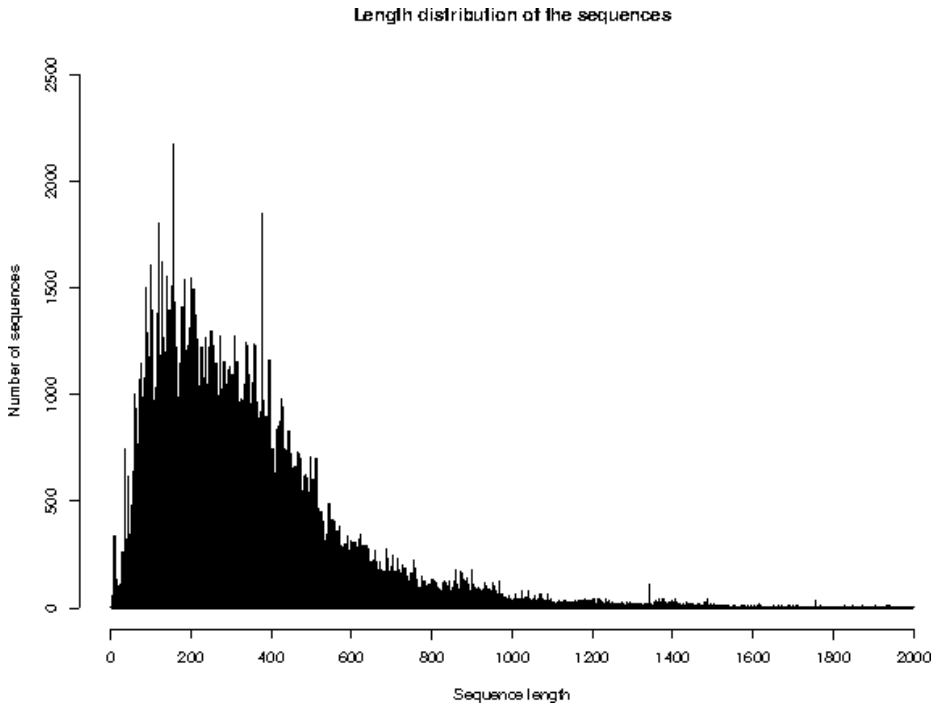


Figure 2.3 : The UniProtKB/Swiss-Prot knowledgebase sequences by length distribution [8]

As of November 2012, the UniprotKB database (Release 2012_11) comprises 538,585 sequences or 191,240,774 amino acids. The average sequence length is 355 amino acids,

with the shortest of 2 amino acids and the longest 35,213 [8]. Performing alignment matrix computation in hardware such as FPGA requires at least the same numbers of processing elements (PEs). This is due to the fact that the PE only capable to hold one residue of a query sequence at a time to compute an alignment score. From hardware point of views, the number of PEs depends on the hardware resources available. In the case of midrange type of FPGA devices, such as the Virtex-5 XC5VLX110, the PE can be reused by computing alignment matrix in several passes, using the so-called folded systolic array architecture.

In addition, the amount of biological sequences in the database increases exponentially over years. As an example, Figure 2.4 illustrates the exponential increase of the UniProtKB/TrEMBL database since the start of Human Genome Project (HGP) circa 1991. This shows the need for high performance computing platforms including processors with multi-core architecture, high performance supercomputers, GPUs and FPGAs to accelerate sequence homology search in order to get results in realistic time.

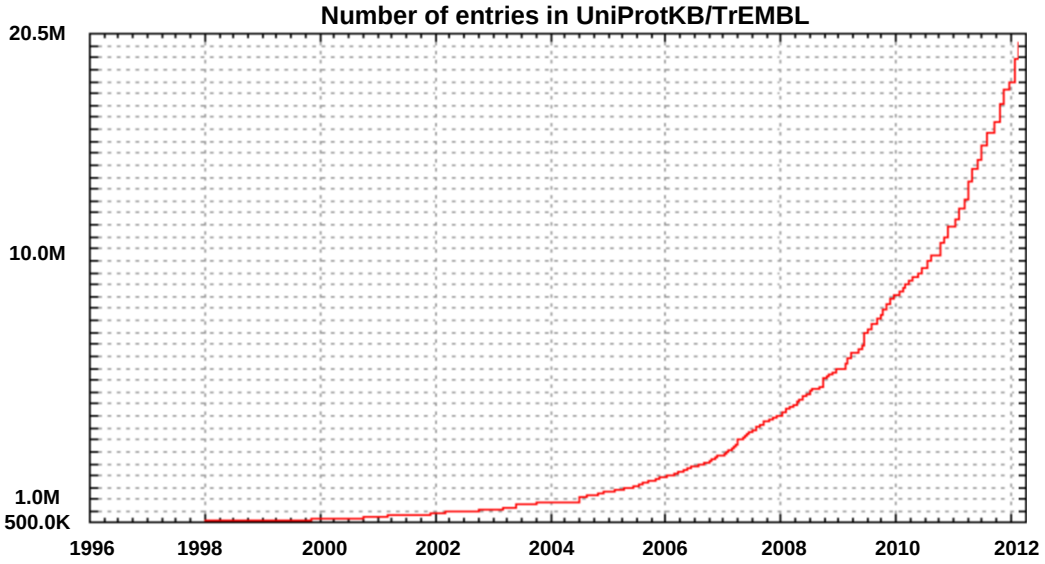


Figure 2.4: Exponential growth of number of biological sequences in the UniprotKB/TrEMBL protein database over years [8]

2.3 Alignment algorithms

Alignment algorithms facilitate the determination of potential sequences in a database to be chosen as biologically related to the query sequence. The algorithms specify scores for various ways to compare pair of sequences in a pairwise sequence alignment. The calculated score of each set of alignment is then used to rank between set of alignments. The underlying formulas behind the alignment algorithms range from the simple sequence edit distance function to a more sophisticated and complex maximum-likelihood values [9]. In the sequence edit distance approach, three scores are specified [9]: (i) score when aligning identical residues (match) of two biological sequences; (ii) the cost when aligning pair two different residues (mismatch); (iii) the cost when aligning a residue in a sequence to a gap in another sequence. As an example, given two short DNA sequences, *x* of length 9 residues and *y* of length 10 residues as in Figure 2.5, performing the sequence alignment of these two sequences may result in different possible sets of alignment, as shown in Figure 2.6.

x : AGGCTAGTT

y : AGCGAAGTTT

Figure 2.5: Example DNA sequences

Now, the sign '-' in Figure 2.6 to denote a space (gap) is introduced between the alignments of the two sequences to potentially extend their positions in a one-to-one correspondence residue from a pairwise sequence alignment.

x		*A*	*G*	*G*	*C*	*T*	*A*	*G*	*T*	*T*	-	
y		*A*	*G*	*C*	*G*	*A*	*A*	*G*	*A*	*T*	*T*	
x		*A*	*G*	*G*	*C*	*T*	*A*	-	*G*	*T*	*T*	-
y		*A*	*G*	-	*C*	*G*	*A*	*A*	*G*	*T*	*T*	*T*
x		*A*	*G*	*G*	*C*	-	*T*	*A*	-	*G*	*T*	*T*	-
y		*A*	*G*	-	*C*	*G*	-	*A*	*A*	*G*	*T*	*T*	*T*

Figure 2.6: Possible alignments for two DNA sequences (*x* and *y*) with *x* as the query sequence and *y* the subject sequence.

The top *x-y* alignment has 6 matches, 3 mismatches and 1 gap. The middle alignment has 7 matches, 1 mismatch and 3 gaps while the last possible alignment has 7 matches, no mismatches and 5 gaps. Alignment score for each of the three possible alignments is required to rank the degree of homology between them. The score is determined by assigning a scoring function to quantify the edit distance in terms of numbers of matches, mismatches and gaps between two sequences as depicted in equation 2.1, where *m* is match score, *s* is a mismatch and *d* is a gap score.

$$Score = m\sum match - s\sum mismatch - d\sum gap \qquad (2.1)$$

In this example, if *m=s=d=1*, then the top *x-y* alignment score is 2. The middle alignment score is 3 followed by 2 for the last one. Note that gaps and mismatches in alignment are undesirable as minimum numbers of changes (insertions and deletions) are expected to convert from one sequence to another. Thus, gaps in an alignment are penalized by subtracting their values from the alignment score as shown in equation 2.1. The degree of homology between the three possible alignments is expressed in the form of overall score following equation 2.1. Based on the given example, the middle alignment clearly produced the best alignment with score of 3.

Early sequence alignment algorithms used this scoring scheme to quantify the degree of similarity between sequences, which is likely to be suitable for DNA sequence alignments. In the case of protein sequence alignments, different scores are required for every substitution of the 20 main amino acids. Therefore a more biologically meaningful approach to quantify the degree of homology between sequences is required. This is realized by the use of probability scores in the form of a score matrix known as substitution matrix to relate biological relationships between amino acids.

2.3.1 Substitution matrices

Substitution matrices specify a probability score when aligning a pair of amino acid residues. The use of a substitution matrix or score matrix in alignment algorithms enables the consideration of biological factors, including the evolutionary histories of biological molecules in alignment algorithms. Score matrix, which models such evolutionary histories are presented in the form of a two-dimensional matrix with each

row and column representing the scores of amino acid residues. In 1978 Dayhoff et al. [10] introduced probabilistic matrices for the substitution of amino acid residues which were known as percent accepted mutation or point accepted mutation (PAM). Later, in 1992, Henikoff et al. [11] proposed another score matrix known as blocks substitution matrices (BLOSUM). Examples of such matrices are PAM 250, PAM 80, BLOSUM 45, BLOSUM 50 and BLOSUM 62. Both PAM and BLOSUM are commonly used substitution matrices in biological sequence alignments. PAM metrics are developed based on global alignments of protein sequences, while the BLOSUM probability scores are based on local alignments. For instance, the BLOSUM 62 is developed from comparisons of sequences with 62 percent identity, while BLOSUM 50 is based on comparisons of sequences with 50 percent identity [12]. These values are referred to as threshold identity. The higher the number, the more highly conserved sequence identities, while the lower threshold gives more divergent identity of the matrix [12]. The BLOSUM 62 matrix has become the *de facto* standard in many sequence alignment programs include in the BLAST algorithm because it is empirically performs very well and suitable for alignments of moderately related sequences [12]. On the other hand, the BLOSUM 50 is widely-used substitution matrix in the SSEARCH program. This software calculates alignment score using the Smith-Waterman algorithm with affine gap penalty. As an example, Figure 2.7 presents the BLOSUM 50 substitution matrix with all entries on the main diagonal highlighted in bold for identical residue pairs.

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	**5**	−2	−1	−2	−1	−1	−1	0	−2	−1	−2	−1	−1	−3	−1	1	0	−3	−2	0
R	−2	**7**	−1	−2	−4	1	0	−3	0	−4	−3	3	−2	−3	−3	−1	−1	−3	−1	−3
N	−1	−1	**7**	2	−2	0	0	0	1	−3	−4	0	−2	−4	−2	1	0	−4	−2	−3
D	−2	−2	2	**8**	−4	0	2	−1	−1	−4	−4	−1	−4	−5	−1	0	−1	−5	−3	−4
C	−1	−4	−2	−4	**13**	−3	−3	−3	−3	−2	−2	−3	−2	−2	−4	−1	−1	−5	−3	−1
Q	−1	1	0	0	−3	**7**	2	−2	1	−3	−2	2	0	−4	−1	0	−1	−1	−1	−3
E	−1	0	0	2	−3	2	**6**	−3	0	−4	−3	1	−2	−3	−1	−1	−1	−3	−2	−3
G	0	−3	0	−1	−3	−2	−3	**8**	−2	−4	−4	−2	−3	−4	−2	0	−2	−3	−3	−4
H	−2	0	1	−1	−3	1	0	−2	**10**	−4	−3	0	−1	−1	−2	−1	−2	−3	2	−4
I	−1	−4	−3	−4	−2	−3	−4	−4	−4	**5**	2	−3	2	0	−3	−3	−1	−3	−1	4
L	−2	−3	−4	−4	−2	−2	−3	−4	−3	2	**5**	−3	3	1	−4	−3	−1	−2	−1	1
K	−1	3	0	−1	−3	2	1	−2	0	−3	−3	**6**	−2	−4	−1	0	−1	−3	−2	−3
M	−1	−2	−2	−4	−2	0	−2	−3	−1	2	3	−2	**7**	0	−3	−2	−1	−1	0	1
F	−3	−3	−4	−5	−2	−4	−3	−4	−1	0	1	−4	0	**8**	−4	−3	−2	1	4	−1
P	−1	−3	−2	−1	−4	−1	−1	−2	−2	−3	−4	−1	−3	−4	**10**	−1	−1	−4	−3	−3
S	1	−1	1	0	−1	0	−1	0	−1	−3	−3	0	−2	−3	−1	**5**	2	−4	−2	−2
T	0	−1	0	−1	−1	−1	−1	−2	−2	−1	−1	−1	−1	−2	−1	2	**5**	−3	−2	0
W	−3	−3	−4	−5	−5	−1	−3	−3	−3	−3	−2	−3	−1	1	−4	−4	−3	**15**	2	−3
Y	−2	−1	−2	−3	−3	−1	−2	−3	2	−1	−1	−2	0	4	−3	−2	−2	2	**8**	−1
V	0	−3	−3	−4	−1	−3	−3	−4	−4	4	1	−3	1	−1	−3	−2	0	−3	−1	**5**

Figure 2.7: The BLOSUM 50 substitution matrix [11]

2.3.2 Gap penalties

A biologically meaningful sequence alignment also considers another important element in calculating alignment scores. These are called as gap penalties. Gaps are undesirable and thus each gap in alignment is penalized with a certain value, known as the gap cost. There are two widely-used gap models in sequence alignment algorithms; linear and affine gap penalty models [3]. A gap cost in the linear gap model is a constant value, but for the affine gap model, the gap cost is determined by a more realistic function. The latter comprises of gap-open (d) and gap-extension (e) penalties. The gap-open is a gap cost used when opening a new gap in a sequence and the subsequent gaps following the initial gap are penalized linearly (gap-extension). A standard gap penalty cost associated with a gap of length g is given either by equation 2.2 for a linear gap penalty or equation 2.3 for the affine gap model. In either model, the value g is dependent on the length of the gap. Further gaps in sequence alignment increase the gap value proportionally.

$$penalty\,(g) = -gd \qquad (2.2)$$

$$penalty\,(g) = -d - (g-1)\,e \qquad (2.3)$$

2.4 Classification of sequence alignment algorithms

The aforementioned biological processes of residue substitution, the insertion of new residue or deletion of an existing residue in a sequence are modeled by mathematical equations or alignment algorithms to quantify the degree of homology between sequences under comparison. Typically, sequence alignment algorithms are categorized into two broad classes; optimal and heuristic search techniques. The former uses the dynamic programming-based algorithm to search for sequence homology and guaranteed to find optimal scores; however, it intensive search technique is time consuming. Examples of the optimal alignment algorithms include the Smith-Waterman algorithm, the Needleman-Wunsch algorithm and the Hirschberg's algorithm. The first two algorithms are the widely-used algorithms, where the first one focuses on local alignment while the second is for global alignment. Alternatively, the heuristics-based approaches produce results faster than the optimal one. However, the advantage of faster matrix filling operations compared to the optimal approaches comes with the

disadvantage of less sensitive alignment scores due to their sub-optimal search techniques. Fast alignment (FASTA) and basic local alignment search tool (BLAST) are examples of the heuristic algorithm. The following sections describe in more details operations of each type of sequence alignment algorithm.

2.4.1 Dynamic programming-based optimal alignment algorithms

The dynamic programming approach solves a complex problem by breaking down the main problem into a reasonable number of simpler sub-problems. Then, these sub-problems are solved recursively, which ultimately give an optimal solution to the main problem. The Needleman-Wunsch and Smith-Waterman algorithms are examples of dynamic programming-based sequence alignment algorithms. The Needleman-Wunsch is a global alignment algorithm introduced by Needleman and Wunsch in 1970[13], while the Smith-Waterman algorithm is a local alignment algorithm proposed in 1981 by T. F. Smith and M. S. Waterman [14]. Given sequence *x* is the query sequence of *'H' 'E' 'A' 'G' 'A' 'W' 'G' 'H' 'E' 'E'* amino acid residues and sequence *y* is the subject sequence of *'P' 'A' 'W' 'H' 'E' 'A' 'E'* amino acid residues, the Needleman-Wunsch performs global alignment of all of the sequences in x and y using equation 2.4.

$$F(i,\ j) = \max\begin{cases} F(i-1,\ j-1) + s(x_i, y_j) \\ F(i-1,\ j) - d \\ F(i,\ j-1) - d \end{cases} \qquad (\ 2.4\ )$$

On the other hand, the Smith-Waterman algorithm searches for the best alignment between sub-sequences x and y using equation 2.5. Both equations are identical, except that zero is added to the maximum expression in the case of local alignment. This ensures that local alignment scores saturate to zero, whereas global alignment scores can take negative values in aligning entire sequences of *x* and *y*.

$$F(i,\ j) = \max\begin{cases} 0 \\ F(i-1,\ j-1) + s(x_i, y_j) \\ F(i-1,\ j) - d \\ F(i,\ j-1) - d \end{cases} \qquad (\ 2.5\ )$$

Since the local and global alignment algorithms are almost identical, the global algorithm with a linear gap penalty, the matrix filling operations as shown in Figure 2.8, is used to describe the matrix filling operation of the dynamic programming algorithms in equation 2.4 and equation 2.5.

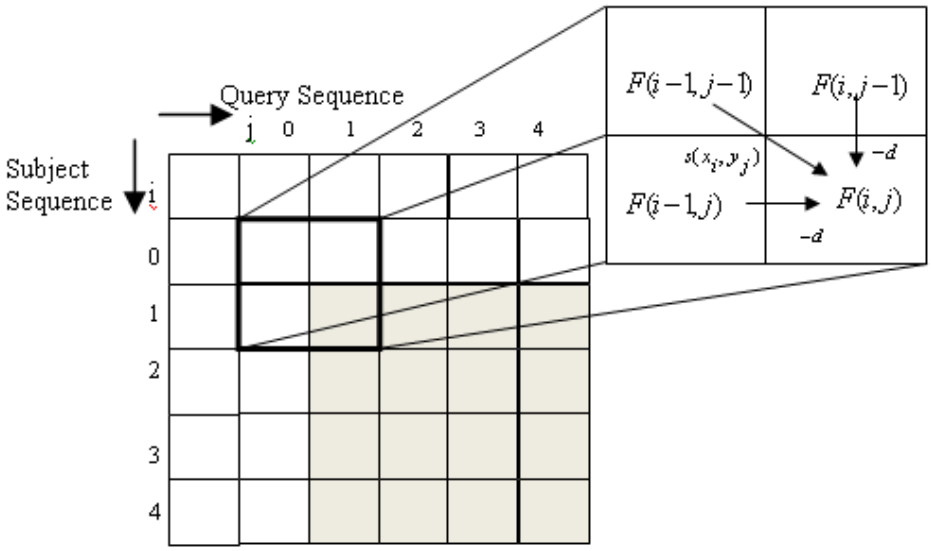


Figure 2.8: Computing *F(i,j)* in an alignment matrix *F*

The alignment matrix *F* is indexed by *i* and *j* with one index per sequence character or residue. The three adjacent elements (diagonal, top and left elements) are used to compute the score of *F(i,j),* whereby the ultimate score for cell *F(i,j)* is the highest score from any of the three possible alternatives;

- diagonal element $F(i-1, j-1)$,
- top element $F(i, j-1)$,
- left element $F(i-1, j)$.

The $s(x_i, y_j)$ is the corresponding substitution matrix score for residue *i* in sequence *x* and residue *j* in sequence *y* respectively. Before constructing the alignment matrix *F* recursively, boundary values of the alignment matrix are required. The *F(0,0),* is set to zero, as it obviously does not represent any alignment either in sequence x or y. It is

thus always set to zero for both global and local alignment. In the case of global alignment, *F(i,0)*, which represents the alignment of prefix *x* to all gaps in *y*, must be set to *-id* . Similarly for the *F(0,j)*, which is set to *-jd* as it represents the alignment of prefix *y* to all gaps in *x* direction. In the case of local alignment, all boundary values (*F(i,0)* and *F(0,j)*) are set to zero since the alignment searches for local similarity between subsequent portions of the two sequences. Computation of the alignment matrix *F* starts from the top left of the similarity matrix, as illustrated in Figure 2.8. This matrix is then built up recursively from the first segment $x_{1\ldots i}$ of x up to x_i and the first segment $y_{1\ldots j}$ up to y_j. In order to illustrate the differences between these two algorithms, a query sequence, *x = 'M' 'E' 'A' 'G' 'H' 'W' 'E' 'E' 'C' 'A' 'M' 'M'* and subject sequence, *y = 'W' 'E' 'E' 'G' 'A' 'A' 'W' 'P'* is use as an example. In the case of the example query and subject sequence, the corresponding alignment matrices for global and local alignment are shown in Figure 2.9 and Figure 2.10 respectively. Note that, in these examples, BLOSUM 50 has been used as the substitution matrix. The bold values in both figures mark the trace back procedure. In the case of global alignment, the trace back starts from the bottom right of the alignment matrix. This alignment algorithm attempts to search homology between entire region of the sequences *x* and *y*. On the other hand, the local alignment algorithm focuses on searching similarity region between the sub-sequence of *x* and *y*. The resulting alignment of the respective sequence alignment algorithm is illustrated at the bottom of each figure.

		M	**E**	**A**	**G**	**H**	**W**	**E**	**E**	**C**	**A**	**M**	**M**
	0	-8	-16	-24	-32	-40	-48	-56	-64	-72	-80	-88	-96
W	-8	**-1**	**-9**	**-17**	**-25**	**-33**	**-25**	-33	-41	-49	-57	-65	-73
E	-16	-9	5	-3	-11	-19	-27	**-19**	-27	-35	-43	-51	-59
E	-24	-17	-3	4	-4	-11	-19	-21	**-13**	-21	-29	-37	-45
G	-32	-25	-11	-3	12	4	-4	-12	-20	**-16**	-21	-29	-37
A	-40	-33	-19	-6	4	10	2	-5	-13	-21	**-11**	-19	-27
A	-48	-41	-27	-14	-4	2	7	1	-6	-14	-16	**-12**	-20
W	-56	-49	-35	-22	-12	-6	17	9	1	-7	-15	-17	**-13**
P	-64	-57	-43	-30	-20	-14	9	16	8	0	-8	-16	**-20**

M E A G H W E E C A M M -
W - - - - - E E G A A W P

Figure 2.9 : Alignment matrix *F(i,j)* for finding optimal alignment using the Needleman-Wunsch alignment algorithm with linear gap penalty ($d = -8$)

		M	E	A	G	H	W	E	E	C	A	M	M
	0	0	0	0	0	0	0	0	0	0	0	0	0
W	0	0	0	0	0	0	0	0	0	0	0	0	0
E	0	0	**6**	0	0	0	0	**6**	6	0	0	0	0
E	0	0	**6**	5	0	0	0	6	**12**	4	0	0	0
G	0	0	0	**6**	**13**	5	0	0	4	**9**	4	0	0
A	0	0	0	5	6	**11**	3	0	0	3	**14**	6	0
A	0	0	0	5	5	4	8	2	0	0	8	**13**	5
W	0	0	0	0	2	2	**19**	11	3	0	0	7	12
P	0	0	0	0	0	0	11	**18**	10	2	0	0	4

E A G H W E E
E - G A A W P

Figure 2.10 : Alignment matrix *F(i,j)* for finding optimal alignment using the Smith-Waterman with linear gap penalty ($d = -8$)

2.4.1.1 Dynamic programming with more accurate models

Both the global and local alignment algorithms presented in the previous section use a linear gap penalty. Another gap penalty which is more accurately models biological processes is known as the affine gap penalty model as introduced in section 2.3.2. This affine gap penalty model was proposed by Gotoh [15] in 1982, which is further improvement of the alignment algorithms with the linear gap penalty model. Unlike the linear gap model, which uses a constant value to penalize an alignment score, the gap cost of the affine gap model uses more realistic approaches. The improved version of the dynamic programming-based alignment algorithms are expressed by equations 2.6, 2.7 and 2.8.

$$F(i,j) = \max\begin{cases} F(i-1,j-1) + s(x_i, y_j) \\ I_x(i-1,j-1) + s(x_i, y_j) \\ I_y(i-1,j-1) + s(x_i, y_j) \end{cases} \qquad (2.6)$$

$$I_x(i,j) = \max\begin{cases} F(i-1,j) - d \\ I_x(i-1,j) - e \end{cases} \qquad (2.7)$$

$$I_y(i,j) = \max\begin{cases} F(i,j-1) - d \\ I_y(i,j-1) - e \end{cases} \qquad (2.8)$$

Generally, optimal alignment guarantees sensitive alignment but with the expense of computational time complexity. With the ever-increasing number and size of biological databases, which have increased exponentially over the years, there is a need for an optimized algorithm to produce results in a realistic time. Alternatively, heuristic-based sequence alignment can be used to get results more quickly, however with less sensitive alignment as compared to the optimal one.

2.4.2 Heuristic-based alignment algorithms

Fast Alignment (FASTA) [16] and Basic Local Alignment Search Tool or BLAST [17] are examples of heuristic-based sequence alignment algorithms. FASTA was developed by Lipman and Pearson in 1985 [16] and was further improved three years later[18]. The need for better search speeds then led to the development of a better algorithm known as BLAST in 1990. Introduced by Altschul et al. [17], the BLAST algorithm searches for a statistically significant alignment from a high scoring pair or HSP of aligned words. Unlike in optimal alignment which calculates an entire alignment matrix, the heuristic algorithm only calculates regions with high scoring pairs. Other regions or insignificant hits that are very far away from the main aligned region in the matrix will be discarded. This reduces the time used to calculate the entire alignment matrix. An illustration of the gapped BLAST with the two-hit method is shown in Figure 2.11.

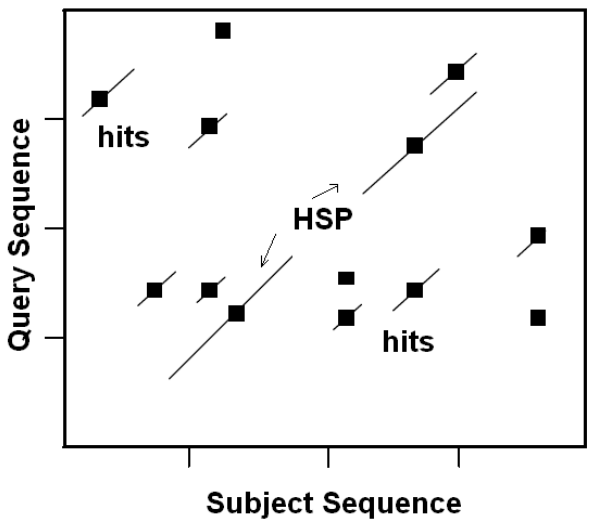


Figure 2.11: Illustration of random and meaningful hits in an alignment matrix

The BLAST algorithm has three important stages. The first creates a list of *W* overlapping words of a given query sequence, where *W* typically 11 residues for DNA sequence alignment and *W* equals to 3 for protein sequence alignment. Then, the pre-processed query words are scored against each subject sequence in the database with aim to search for meaningful hits of the generated words. This stage is referred to as the seed generation stage and during this stage, there are two types of hits generated; random hits and meaningful hits. Random hits, labeled as hits as in Figure 2.11 are undesirable, thus these hits are filtered out leaving only meaningful hits (known as seeds or high scoring pairs (HSPs)) as in Figure 2.11. Scores of these seeds are then calculated in the second stage of BLAST algorithm known as the ungapped extension stage. Lastly, those seeds with ungapped alignment scores satisfying a given threshold value is extended in the gapped extension stage. Details of each stage are discussed in Chapter 6.

2.4.3 Profile HMM sequence alignment

The theory of hidden Markov models (HMMs) has been used in speech recognition for years and it is now applied in molecular biology due to its suitability for 'linear' problems such as sequences and time series [19]. A profile HMM is essentially a probabilistic model that represents the positions specific of highly conserved sequence patterns or motifs in multiple sequence alignment. Motifs exist in evolutionary-related sequences. Mutation, selection, and genetic drift create variations of biological sequences from their common ancestor. These manifest themselves as residue substitution, deletion or insertion. A profile HMM is modeled using discrete states, where each state represents motif positions with probability scores assigned to the state and its transitions. To understand this representation, one can imagine that an HMM generates a certain sequence [20]. When a state is visited, a residue is emitted from the state based on an emission probability score. The transition from state to state generates the underlying state path, which is referred to as a Markov chain. Figure 2.12 illustrates an example of three consensus columns of a multiple sequence alignment of five sequences. The consensus column or motif is assigned to its corresponding match state (*M*). Each match state has an emission probability as it is visited by subject sequence residues in that position. Each match state is accompanied by the other two states of insertion (*I*) and deletion (*D*) states. An insertion state exists between two match states

and each insertion state has a state transition to itself which allows for the insertion of one or more residues and a next match state. The delete state is a 'mute' state since it emits nothing when a subject sequence residue visits this state. This reflects residue deletion [19] in the biological process. To evaluate the probability of a sequence being generated by the model, a dynamic programming-based algorithm called the Viterbi algorithm is used. Details of hidden Markov-based sequence alignment are presented in Chapter 5.

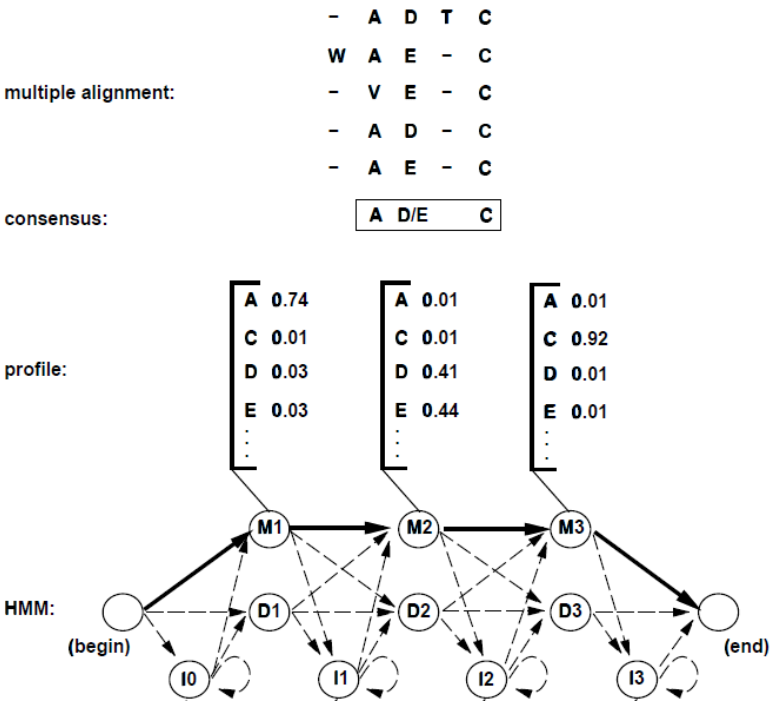


Figure 2.12: Consensus columns of a multiple sequence alignment and their corresponding profile HMM [21].

2.5 Summary and conclusions

In this chapter, the essential background of biological sequence alignment has been presented. Sequence alignment is a fundamental tool in molecular biology which aims to find regions of similarity between sequences. Among other applications, sequence alignment is useful in early disease diagnosis, drug engineering and in facilitating the construction of phylogenetic trees. Sequence homology involves the search for the best matched sequences in a genomic database. Scores are used to represent the degree of homology between the searched sequences, and this is done through the matrix filling operations. The score of each cell in an alignment matrix is calculated using alignment algorithms, and there are two broad classes of these algorithms. The first one is the optimal alignment, while the second one is the heuristic-based approaches. The former are guaranteed to find optimal results through its intensive dynamic programming-based search algorithms but this approach requires quadratic time complexity when run on a standard microprocessor, while the latter produce results in a realistic time with disadvantage of less sensitive alignment. The Smith-Waterman and the Needleman-Wunsch algorithms are examples of such optimal alignment algorithms, while FASTA and BLAST are examples of heuristic-based sequence alignment algorithms. On the other hand, the profile HMM-based sequence alignment is another type of sequence alignment, which falls under the heuristic-based sequence alignments. It adopts the theory of hidden Markov to model the positions specific of multiple sequence alignments. The profile-to-sequence alignment is performed using the same mechanism as that in pairwise sequence alignment, whereby the profile HMM is searched against each subject sequence in a database.

In this research, the Smith-Waterman with affine gap penalty or sometimes referred to as the Gotoh algorithm is chosen as it is the most sensitive algorithm for the pairwise sequence homology search. Due to its intensive search, this optimal alignment algorithm requires proportional amount of time as the number of sequences in a genomic database increase over years. Alternatively, the gapped BLAST with the two-hit method will be implemented. BLAST is chosen rather than FASTA due to BLAST offers more sensitive and faster search as compared to other heuristic-based algorithms. In addition, profile HMM-based sequence alignment will be implemented due to its powerful homology search technique as it is able to search for remotely homologous sequences.

Chapter 3
Introduction to Field Programmable Gate Arrays

This chapter discusses the historical background of FPGAs and explains the generic structure of FPGA fabric. Then, the performance of FPGAs compared to other computing platforms is elaborated. Following this, the CAD tools available for FPGA-based designs are described, and the Alpha Data board used in this research is introduced before summary and conclusions of this chapter are laid out.

3.1 The emergence of the FPGA

The first programmable logic device (PLD) specifically used for implementing logic circuits was the programmable logic array (PLA). It was introduced by the Philips company in early 1970s [22]. The PLA consists of a programmable AND-plane followed by a programmable OR-plane. However, the use of these two levels of configurable logic resulted in poor speed performance and expensive manufacture. The next generation of PLDs known as programmable array logic or PAL was introduced to overcome the aforementioned issues. Unlike PLA, PAL had only a single level of programmability (a programmable AND-plane followed by a fixed OR-plane). Several variants of basic PAL architecture were then produced to compensate for the lack of generality due to the fixed OR-plane. To overcome their limitations, PLA and all PAL variants were then grouped together. Combinations of such programmable logic devices produced simple PLDs (SPLDs) at low cost and with very high pin-to-pin speed performance. Subsequent advances in fabrication technology closely followed the Moore's law curve, where the number of transistors approximately doubling every two years, enabling fabrication of PLDs with multi-level logic architecture and integration of these SPLD-like blocks (referred to as macrocells) onto a single chip with programmable interconnections between each of the blocks. This type of device had a large logic capacity to allow the implementation of more complex design and thus referred to as complex PLDs (CPLDs). CPLDs were first introduced by Altera in their family known as Classic EPLDs followed

by three other series of CPLDs; the MAX 5000, MAX 7000 and MAX 9000 [22]. During that time, the logic capacity of CPLDs was extendable to a maximum of about 50 typical SPLD device [22]. For higher logic capacity, other approaches such as the Mask-Programmable Gate Array (MPGAs) were used. However, this general purpose logic chip required large setup costs in customizing the user's logic circuit during fabrication and a long manufacturing time was required. This led to the use of another user-programmable equivalent, known as Field Programmable Gate Arrays (FPGAs), since about 1986 [23]. An FPGA comprises of an array of configurable logic blocks and interconnections which can be configured an infinite number of times in the 'field' (i.e. through programming) by the end user. FPGAs can be programmed either by anti-fuse, flash or SRAM-based programming methods. Table 3.1 summarizes the characteristics of each way to configure FPGA with configuration bitstream.

Table 3.1: Different FPGA programming methodologies [24]

Criteria	Anti-fuse	FLASH/EEPROM	SRAM
Reprogrammable	No	Yes	Yes
Volatile	No	No	Yes
In-System Programmable (ISP)	No	Yes	Yes
Area	Low	Moderate	high
Manufacturing Process	Anti-fuse	Flash	Standard CMOS
Programming Yield	>90%	~100%	~100%

Among these programming methods, the SRAM-based method is the most widely-used as it uses a standard SRAM cell to store configuration data. This allows such SRAM-based FPGAs to benefit from the advantages of the latest CMOS technology. CMOS technology offers higher speed performance and lower dynamic power consumption with increased integration onto a single silicon chip. Although this technique requires more area (one SRAM cell comprises either 5 or 6 transistors) and cannot retain configuration bits when the power is turned off i.e. it is volatile, the SRAM cell could be programmed an indefinite number of times. In addition, the SRAM does not require special integrated circuit processing steps in order to retain data while in operation.

The first modern FPGA was introduced in 1984 by Xilinx [24]. It comprises of a classic array of configurable logic blocks. During that time FPGAs contained 64 logic blocks and 58 I/Os [24]. Nowadays, FPGAs have grown enormously in complexity offering, for example, up to 305,400 slices and around 1200 user I/Os for the Virtex-7 FPGA (XC7V2000T) [25]. Moreover, significant architectural changes, such as large numbers of more specialized blocks, have been introduced in modern FPGAs, including embedded DSP slices, blocks RAM, embedded microprocessors, and high speed I/O buses. These additional resources and the higher logic density offered by modern FPGAs not only increase their usage in more complex designs but also have greatly expanded the capability to become viable alternatives in high performance scientific computing.

3.2 Architecture of modern FPGAs

Altera and Xilinx are the two market leaders in FPGA manufacturing industry. Other FPGA manufacturers include Atmel, Cypress, Lattice Semiconductor and Actel. FPGAs are semiconductor devices which mainly comprise of three main elements; programmable logic blocks, programmable routing interconnects, and interfaces to the external connections of the device [26]. The generic architecture of modern FPGAs comprises general logic blocks i.e. configurable logic blocks (CLBs) for Xilinx FPGAs and adaptive logic modules (ALMs) for Altera FPGAs, DSP blocks and embedded memory (the block RAM) as shown in Figure 3.1.

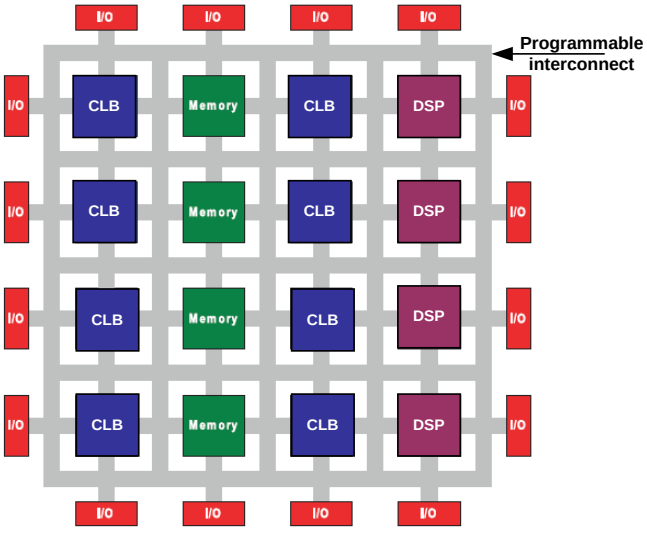


Figure 3.1: Internal structure of Xilinx FPGA [24]

Each of these block is surrounded by programmable routing fabric (programmable interconnect) that allows the connection of the blocks to implement more complex circuitry. This is realized by configuring the necessary connections using the programmable interconnects between the logic blocks. Interfacing to the outside world is performed through the I/O blocks. I/O blocks are arranged in a ring form around the circumference of the device. Today's FPGAs have additional features including hard core processors (the Power PC for Virtex-II Pro, Virtex-4 FXT and Virtex-5 FXT FPGAs) and soft core processors in the FPGA fabric. This gives designers more design flexibility to achieve better trade-offs between development time, system performance and costs. The area of an FPGA is dominated by its configurable elements, which mainly comprises of arrays of configurable logic blocks. Given the re-programmability feature of the digital integrated circuit, new circuitry can be implemented on the configurable fabric with a tremendous variety of tasks possible on the same silicon chip. Xilinx has two main FPGA families, the Virtex and the Spartan series. Virtex FPGAs have the lowest power and highest performance, while the Spartan series are low cost and high volume FPGAs. Both of these groups are SRAM-based re-programmable logic devices.

3.2.1 Programmable logic blocks

Programmable logic blocks are the main resources in FPGA for the implementation of both non-clocked-based logic (combinatorial logic) and clocked-based logic (synchronous logic) designs. The internal structures and terminology used to define programmable logic blocks vary depending on the FPGA vendors. For instance, Xilinx defines such a logic block as a configurable logic block (CLB), while Altera refers to it as a logic array block (LAB). In the Xilinx FPGA family, a CLB is made up of either two or four logic slices. Table 3.2 summarizes the internal elements in a single CLB in the Virtex family. Early generations of Xilinx devices up to Virtex-4 had four slices (two SLICEL and two SLICEM) as shown in Figure 3.2. SLICEM elements are arranged in the left column, while SLICEL are in the right column of the CLB. Each SLICEM and SLICEL has an independent carry chain. Each slice supports the implementation of arithmetic functions such as addition or subtraction. For wider LUT inputs, the function generators can be cascaded within the CLB or between neighboring CLBs using the switch matrix for coarse grain logic implementation.

Table 3.2 : A single CLB resources in various Virtex family extracted from vendor's user guides

Device	**Virtex-II** [27]	**Virtex-II -Pro** [28]	**Virtex-4** [29]	**Virtex-5** [30]	**Virtex-6** [31]	**Virtex-7** [25]
Slices(#)	4	4	4	2	2	2
LUTs (#)	8	8	8	8	8	8
FFs (#)	8	8	8	8	16	16
Max Distributed RAM* (Bits)	128	128	64	256	256	256
Shift registers* (Bits)	128	128	64	128	128	128
DSP** Slice	Embedded Multiplier	Embedded Multiplier	DSP48	DSP48E	DSP48E1	DSP48E1

* SLICEM only ** DSP slices are not within a CLB, they have dedicated columns of slices for DSP

For any DSP-based hardware realization, dedicated DSP slices can be used. Xilinx incorporates embedded multipliers in the Virtex-II and Virtex-II Pro. Then, beginning from the Virtex-4, dedicated DSP slices are allocated in the FPGAs.

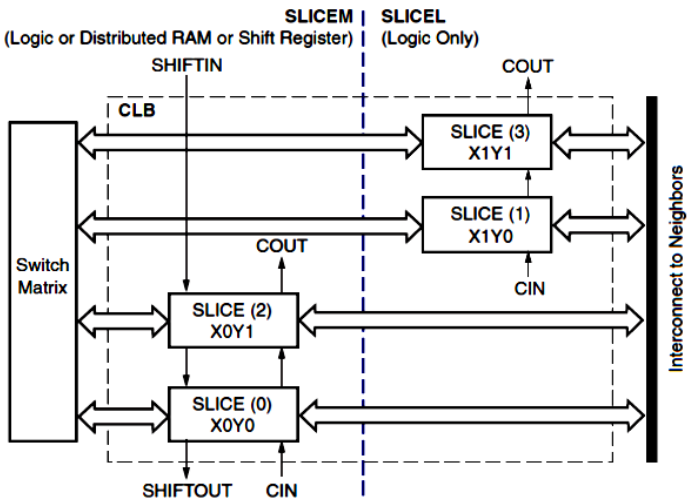


Figure 3.2: Arrangement of slices in single CLB for Virtex-4 and its predecessors [32]

The most primitive part of the CLB architecture is the logic cell (LC) for Xilinx or logic element (LE) for Altera. Among other elements, the logic cell comprises a LUT, a multiplexer and a register, as illustrated in Figure 3.3. Apart from operating as a function

generator, the LUT can also be configured as 16x1 distributed RAM or as a 16-bit shift register. Similarly, the flip-flop can also be configured as a latch.

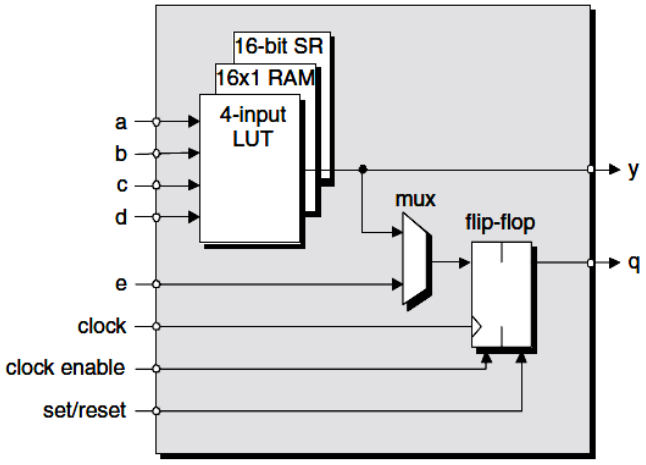


Figure 3.3: Simplified internal architecture of a logic cell in Virtex FPGA [33]

Beginning from the Virtex-5 family, the composition of CLB elements in Xilinx FPGAs is slightly different, whereby the number of inputs for a look-up table is 6-input LUT compared to the 4-input LUT in its predecessors. Each CLB consists of only two slices; SLICEM and SLICEL, as shown in Figure 3.4. These two slices are independent of each other and are organized in different columns with independent carry chains.

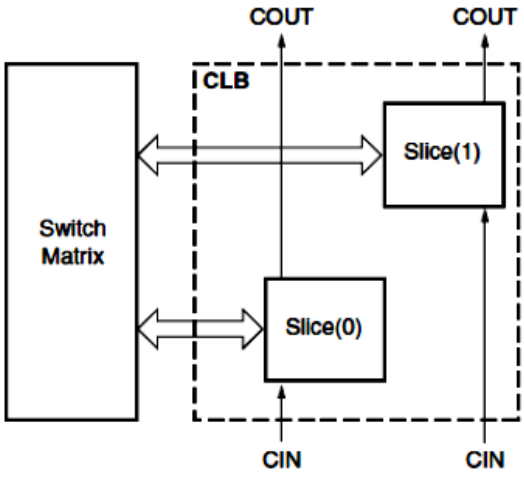


Figure 3.4: Arrangement of slices with single CLB for Virtex-5 FPGA or higher [30]

Among its other advantages, the 6-input LUT offers higher logic density compared to the previous 4-input LUT FPGAs [32]. For instance, to implement a 4-input LUT, 16-bit of memory (in LUT bits) is required. With the advantage of 6-input LUTs, each slice can implement up to 64 bits which represent four-fold logic density. A LUT in SLICEM can also be configured as 64x1 distributed RAM or a 32-bit shift register without using the flip-flops available in a slice [30].

3.2.2 Embedded RAMs

Since most applications require faster memory access, modern FPGAs incorporate embedded memory known as block RAM. Section 3.2.1 has discussed the distributed memory which can be configured using the look-up table in the SLICEM of a CLB. Block RAM (BRAM) is another type of memory embedded in FPGA fabric. Block memory was first used commercially in the Altera Flex10K series FPGA [24]. The BRAM is an on-chip static RAM that offers high speed and customizable memory. The width and the depth of the configurable memory can be set to be narrow or wider depending on the designer's requirements. This memory is not as large as off-chip dynamic memory (DRAM), and therefore it is suitable as a buffer and for local data usage. Nowadays, all contemporary FPGAs have memory blocks in the die area and this trend is likely to continue as on-chip memory elements become crucial. Moreover, most modern FPGAs use memory blocks with dual-port functionality. Such dual-ported memories allow for simultaneous read and write operations, while others allow combinations of both read and write operations.

3.2.3 Embedded multipliers and DSP slices

Modern FPGAs also come with dedicated hard IP resources such as embedded multipliers in the Spartan-3 series, Virtex-II and Virtex-II pro FPGAs. Later, a more sophisticated DSP slices have been embedded in the FPGA fabric for all Spartan-6 and other Virtex families beginning from Virtex-4. Typically, a DSP slice comprises of all of DSP operators required for DSP-based applications, such as multipliers, adders and subtractors. In addition, DSP functions require frequent access to embedded RAM, and thus all of the DSP slices are physically embedded next to the block RAM in order to

give higher computation performance [33]. Figure 3.5 illustrates embedded multipliers in the reconfigurable fabric.

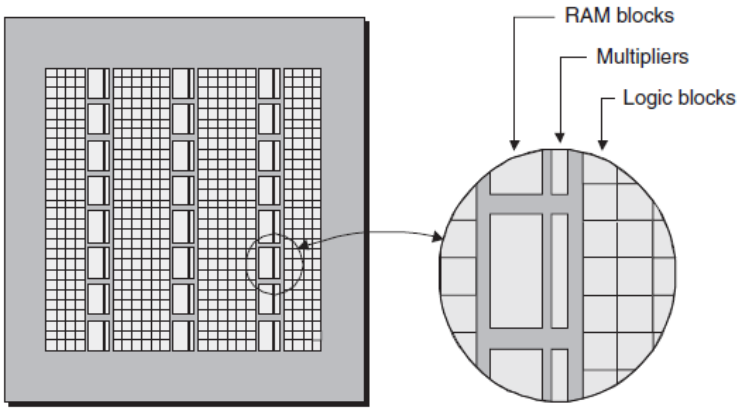


Figure 3.5: Illustration of embedded multipliers and blocks RAM in FPGA [33]

3.2.4 Embedded processors

With the increasing numbers of logic gates in FPGA devices, FPGA vendors integrate embedded microprocessor cores in their silicon chips to enable more design flexibility in order to achieve better trade-offs between development time, performance and cost. Embedded processors in FPGAs can be categorized into two distinct groups; hard microprocessor and soft microprocessor cores. The hard processor is a physical processor core which is permanently embedded in the dedicated silicon area of an FPGA. Examples of this hard IP processor is the ARM922T in the Altera Excalibur family and PowerPC 440 in the Xilinx Virtex-5 FX families, as shown in Figure 3.6.

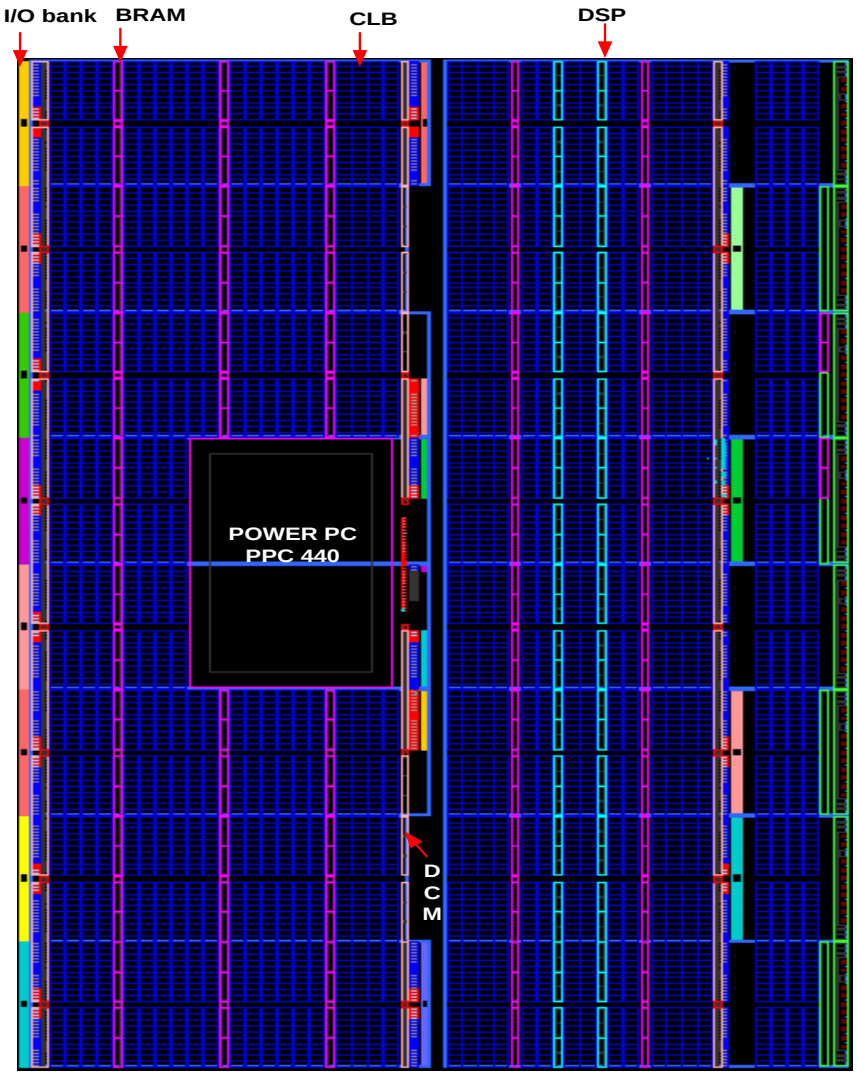


Figure 3.6: A hard IP processor, PPC 440 in the Xilinx Virtex-5 (XC5VFX70T-3FF1136) generated from the Xilinx Plan Ahead tool

Conversely, the soft processor is configured from the FPGA's logic fabric. The soft-processor core or soft IP offers design flexibility, whereby specific peripherals can be customized based on the designer's requirements. Typically, this kind of processor is designed using a netlist generated from a processor design wizard in the Xilinx Platform Studio (XPS) tool suite for Xilinx FPGAs or SOPC Builder for Altera FPGAs. The Altera NIOS II and Xilinx MicroBlaze are examples of such 32 bit RISC processors. Among the advantages of having an embedded processor in an FPGA is that it enables the implementation of high performance embedded applications and offers more design

flexibility. A summary of resources available in the Xilinx Virtex family is presented in Table 3.3.

Table 3.3: Resources available in various Xilinx Virtex FPGAs extracted from vendor's user guides

Device	**Virtex-II** [27]	**Virtex-II -Pro** [28]	**Virtex-4** [29]	**Virtex-5** [30]	**Virtex-6** [31]	**Virtex-7** [25]
Logic Cell	576-104K	3K-125K	13-200K	32K-331K	74K-758K	326K-2M
BRAM	4-168	12-556	48-1.3K	84-516	156-1K	795-1.9K
Dedicated Multiplier	4-168	12-556	N/A	N/A	N/A	N/A
DSP Slice	N/A	N/A	32-512	24-1K	288-2K	1.1K-3.6K
Soft-Processors	The MicroBlaze 32-bit Soft IP Processor core supports for Spartan-3, Spartan-6, Virtex-4, Virtex-5, Virtex-6, Virtex-7 FPGAs [34]					
	The MicroBlaze 8-bit RISC Harvard Soft IP Processor core can be instantiated and supported for all FPGA Family					
Hard-Processor	N/A	IBM PPC	PPC405*	PPC440*	N/A	N/A
DCMs	4-12	4-12	4-20	4-12	6-18	12-24
I/Os	88-1.1K	204-1.2K	320-960	172-1.2K	360-1.2K	600-1.2K

*Virtex-4 FX family and Virtex-5 FXT family only

3.3 Mapping algorithms onto the FPGA

In general, FPGA can be designed from six different design entries; schematic entry, hardware description language (HDL), structured hardware design (hardware skeleton-based architecture), graphical-based design (such as the Matlab System Generator), object oriented design (such as Java-based FPGA design by the Maxeler Technologies) and high level language (such as the Vivado HLS by the Xilinx Corporation). The choice of the design methodologies greatly affected the development time and performance of the design. HDL-based design generally offers more efficient hardware resources utilization due to the design is captured at lower abstraction level, whereas the HLL-based design enables faster time-to-market. In either case, computer-aided design (CAD) tools synthesize the user design into hardware; where complex hardware design is implemented by the collection of a number of logic blocks in an FPGA. Table 3.4 summarizes the most widely-used CAD tools and their respective functionality.

Table 3.4: Computer-aided design tools for FPGAs

Design Stage	Tool	FPGA/EDA vendor
Design entry	ISE	Xilinx
	Vivado HLS	Xilinx
	Quartus	Altera
	FPGA Advantage	Mentor Graphics
Simulation	ISIM	Xilinx
	ModelSim	Mentor Graphics
	Verilog-XL	Cadence
	VCS	Synopsis
	NCSim	Synopsis
	Active-HDL	Aldec
Synthesis	ISE	Xilinx
	Vivado HLS	Xilinx
	Leonardo Spectrum	Mentor Graphics
	Precision RTL Synthesis	Mentor Graphics
Implementation	ISE	Xilinx
	Quartus	Altera
Design Analysis	Plan Ahead	Xilinx
	Chip Planer	Altera
On chip Debugging	Chipscope Pro	Xilinx
	Signal Tap	Altera

Figure 3.7 shows a simplified diagram of a standard FPGA design flow. The design entry involves designing logic circuitry using hardware description language (HDL) either using Verilog or VHDL. The next step is behavioral simulation, which verifies both syntax and functionality of the design before specific netlist files of the design called Native Generic Circuit (NGC) files are generated during the design synthesis stage.

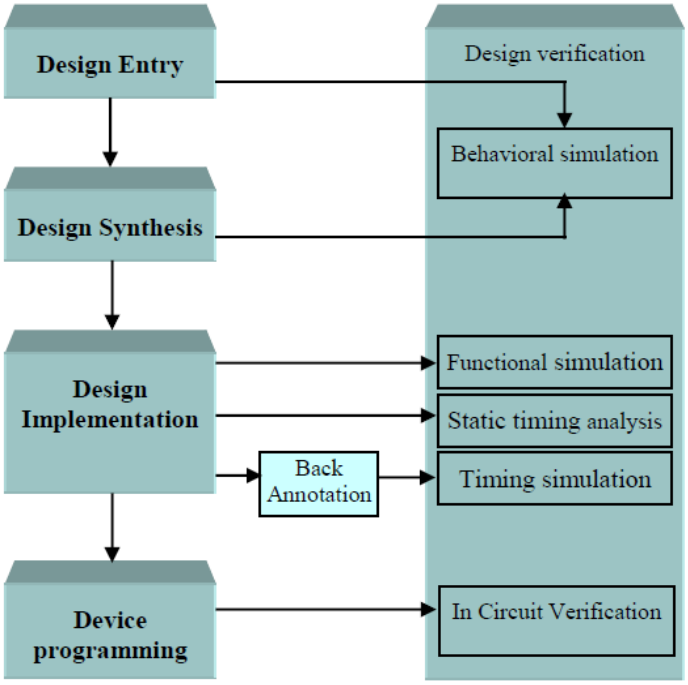


Figure 3.7: The simplified Xilinx FPGA design flow [35]

Once the netlist files are ready, the design implementation stage starts. This stage consists of three sub-stages; Translate, Map, and Place and Route (PAR). Translate uses the NGC files generated in the previous stage to merge the netlist and constraints into a Xilinx design file known as the Native Generic Database (NGD) file. This file may be used for functional simulation to verify the functionality of the design after it is translated. This post-translate simulation stage is optional and it is typically used for debugging translation-related issues. Once the translate step is completed successfully, then the design is mapped into the resources available in the targeted device. This stage generates a Native Circuit Description (NCD) file for the PAR step and the static timing analysis. Static timing analysis evaluates the timing performance of the post-map logic

paths. It creates a post-map static timing report which details the timing information of the logic paths with respect to the requirements of the design and the target device. During this stage, route delays have not yet been reported. The PAR stage places and routes the design following the constraints set in the User Constraint File (UCF). The place and route process generates the static timing report. This report is crucial as it reports whether the design has met the timing requirements or not. Designs with higher levels of logic may cause violations of timing requirements and redesigning the logic paths with fewer logic levels usually solves most timing violation issues. Finally, if the previous stages are successful, the machine code is generated at the device programming stage. The machine code generated can be a partial or a full bit stream file. Configuring FPGAs with full bit files is referred to as static FPGA configuration. On the other hand, bit stream can be partially reconfigured using the partial bit file. The following section discusses in more detail these types of reconfiguration models.

3.4 FPGA reconfiguration models

Static configuration is the most common approach to implement applications onto FPGAs. This type of compile time configuration [36] requires the operation of the target FPGA to be halted, while the entire chip is being reconfigured with new bit streams. The operation then restarts when the configuration process has finished. To alter a subset of data, the entire configuration data needs to be updated onto the FPGA chip and hence this incurs additional time overheads. Nowadays, FPGA technology has advanced so as to enable the reconfiguration of a partial area of an FPGA. This way, the limited resources in an FPGA are time-multiplexed to suit all of the virtual resources required for a particular application. Reconfiguration of a partial area in the FPGA can either be implemented through partial reconfiguration or dynamic partial reconfiguration. The former modifies a subset of configuration data without reconfiguring the whole chip. In this approach, operation in the FPGA remains halted while modification is in progress. In the latter approach a portion of the configuration data is changed without interrupting FPGA operation. As its name implies, this dynamic partial reconfiguration re-allocates hardware dynamically at run time by swapping in and swapping out actual hardware resources as required [36]. This approach is sometimes referred to as dynamic partial reconfiguration (DPR) or run time reconfiguration (RTR). An example of a dynamic

partial reconfiguration operation is illustrated in Figure 3.8. In this example, the FPGA has both static logic and reconfigurable logic regions, as shown in Figure 3.8 (a). The static region is operated without interruption during the reconfiguration of partial bit files. Hardware resources in the FPGA are reconfigured with different partial bit files (A1.bit, A2.bit, A3.bit and A4.bit) based on the application's requirements.

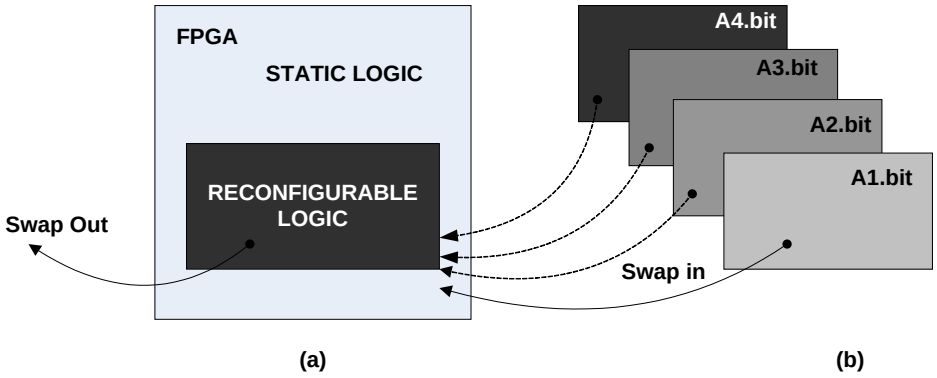


Figure 3.8: (a) Example of static and reconfigurable logic regions in an FPGA (b) Partial configuration bits for different hardware implementations [37]

Reconfiguration of a partial bit stream can be performed either in the self-reconfiguration mode using the Internal Configuration Access Port (ICAP) or using the externally-reconfigurable mode via the JTAG port as illustrated in Figure 3.9. The self-reconfiguration mode can be implemented using an on-chip state machine, processor or other logic, while an off-chip microprocessor or other controller can be used in the case of the externally-reconfigurable mode.

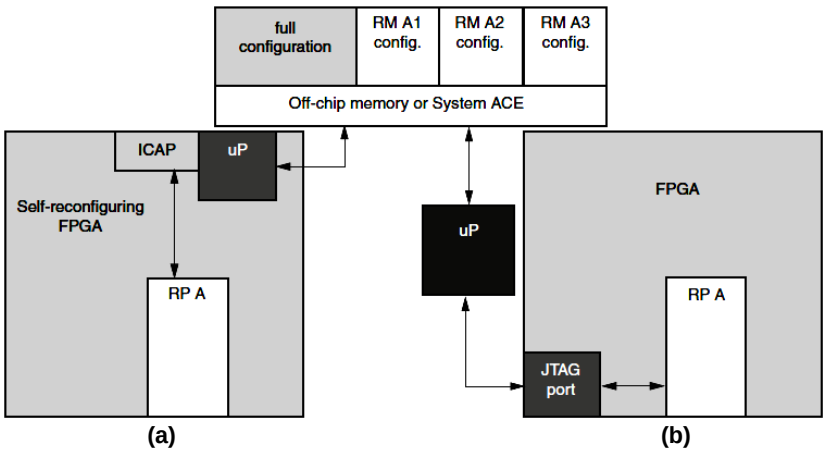


Figure 3.9: (a) Self-reconfiguring mode (b) Externally-reconfigurable mode [38]

The JTAG configuration port is a one bit serial configuration port suitable for applications with no speed urgency. On the other hand, the configuration of partial bit streams through ICAP is faster as it has a data transfer port of 32 bits wide. The self-reconfiguration mode requires the ICAP primitive to be instantiated in HDL description. In either case, the controller retrieves the partial bitstream from the off-chip memory or the System ACE and delivers it to the configuration port (ICAP or JTAG). On top of that, the initial configuration in the static region must be configured onto the target FPGA prior to the reconfiguration of any partial bitstream. As for the static region in DPR, the control circuitry for partial reconfiguration remains uninterrupted during the reconfiguration process. In the case of configuration using the JTAG port, the Xilinx iMPACT tool can be used to load full and partial bitstreams into the target FPGA. The partial bit file can be transferred to FPGA using Slave SelectMAP, Slave Serial, JTAG, ICAP or PCAP. Table 3.5 gives detailed descriptions of each port. Among the advantages of partial reconfiguration is that it reduces the size, power consumption and cost of the FPGA device in implementing a given function. It also optimizes hardware resource utilization by time-multiplexing a reconfigurable region in an FPGA to suit different hardware implementations.

Table 3.5: Configuration ports for partial reconfiguration [38]

Configuration port	Details
ICAP	Suitable for speed demanding applications and it requires instantiation of an ICAP controller and extra logic for ICAP interface.
PCAP	The parallel configuration access port supports all zynq-7000 designs, whereby the processor subsystem (PS) manages the partial reconfiguration operation.
JTAG	JTAG is serial interface, which transfers data using 1-bit signal and it is more suitable for testing and debugging purposes.
Slave SelectMAP or Slave Serial	This configuration port is typically used for full or partial reconfiguration over the same interface port.

Typically, FPGA architectures have configuration memory, which is arranged in frames to manage the configuration of an FPGA device. It manages all other aspects of FPGA design, including signal routing and the LUT equations [38] of the target FPGA. Partial

reconfiguration allows at least a frame of the FPGA elements to be partially reconfigured. A frame is essentially the smallest configuration region in the FPGA for performing partial reconfiguration. Virtex FPGAs support partial configuration of bitstreams, however, Spartan FPGA devices are not supported for this type of FPGA configuration. The following section discusses more details regarding the sizes of frames in various virtex FPGAs.

3.4.1 Configuration frames

The size of the minimum reconfiguration frame varies depending on the FPGA's family and Table 3.6 summarizes the minimum base regions for various Virtex families.

Table 3.6: Minimum sizes of reconfiguration frames for Virtex FPGAs [38]

Device	Minimum frame size
Virtex-4	16 CLBs by 1 CLB wide
Virtex-5	20 CLBs by 1 CLB wide
Virtex-6	40 CLBs by 1 CLB wide
7-Series	50 CLBs by 1 CLB wide

The sizes of the above-mentioned base regions can be determined using the Xilinx Plan Ahead floor planning software [38]. In the case of Virtex-4, Virtex-5 and Virtex-6 FPGAs, the numbers of elements for a reconfiguration frame are summarized in Table 3.7. For the 7-series FPGAs, no information regarding the number of DSPs, BRAMs and IOBs found in literature.

Table 3.7: Elements inside a single reconfiguration frame of Virtex-4 , Virtex-5 and Virtex-6 FPGAs [39].

Device Family	CLB (#)	DSP(#)	BRAM(#)	IOB (#)
Virtex-4	16	8	4	16
Virtex-5	20	8	4	40
Virtex-6	40	16	8	80

3.4.2 Configuration time and considerations for DPR

The size of the configuration frames as discussed in the previous section dictates the time required to partially configure FPGA with partial bitstream. In the case of biological sequence alignment, substitution matrix coefficients inside the processing elements in a systolic array can be updated by reconfiguring a small portion of FPGA area. However, the smallest portion for reconfiguration is the minimum reconfiguration frame. Therefore, reconfiguration is worthwhile if significantly smaller overhead time is required to configure the PE with new coefficients as compared to re-use the PE in folded systolic array architecture. In addition, the configuration time also depends the bandwidth of the reconfiguration port [38]. The Internal Configuration Access Port (ICAP) or Select MAP is suitable for applications where speed is the main consideration, and both of these configuration ports support a maximum data transfer rate of 3.2 Gbps. For example, to reconfigure a small partial bit file of 236,160 bits of a size of 200 slices or 100 CLBs (i.e. 20 CLBs high by 5 CLBs wide on Virtex FPGA), the time required for such operation if the bit file is to be reconfigured using the ICAP or Select MAP is 73.8 microseconds where the 236,160 bits divided by 3,200,000,000 bps [38].

Table 3.8: Various configuration ports in Virtex family [38]

Configuration Mode	Max Frequency (MHz)	Data Width	Max Data Transfer Rate
ICAP	100	32	3.2 Gbps
Select MAP	100	32	3.2 Gbps
Serial Mode	100	1	100 Mbps
JTAG	66	1	66 Mbps

The use of partial reconfiguration methodology in biological sequence alignment has been reported in 2005 by Oliver et al. [40]. In this work, the number of PEs was customizable depending on the length of the query sequence. For efficient hardware utilization, the run time reconfiguration method was used to reconfigure the exact number of PEs based on the length of the query parameters. Unfortunately, no information was given regarding the configuration port used and it was reported that the

reconfiguration time was 80 ms in the case of implementation on the XC2V6000 device. This shows that, although partial reconfiguration is able to make better use of the hardware resources available, the consideration of such an approach in sequence alignment applications is worthwhile only if the PE reconfiguration time is smaller than the time required to perform the matrix fillings operation of an alignment matrix. The subsequent section discusses justifications of choosing FPGA in this work as acceleration platform for biological sequence alignment.

3.5 FPGA performance

Figure 3.10 shows the performance of FPGA and other computing platforms in terms of efficiency (performance, area and power) versus flexibility. The FPGA is the second best in terms of efficiency compared to the application specific integrated circuit (ASIC). However, despite offering the highest efficiency, the non-reprogrammable nature of the ASIC has made such integrated circuits less attractive, especially for high performance computing applications. Moreover, ASIC-based applications also involve longer time to market and production costs are higher if manufactured in small volumes.

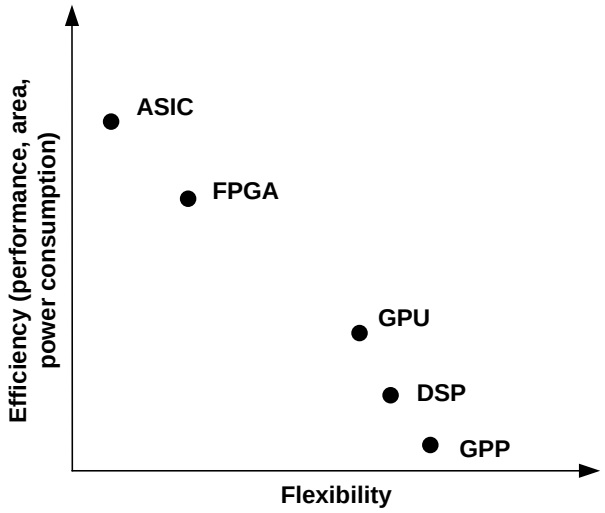


Figure 3.10: FPGA performance against other computation platforms [36]

On the other hand, the flexibility and re-programmability of FPGA comes with relatively higher power consumption as compared to ASIC. This is because the FPGA is not highly optimized for a specific application since its internal architecture is generic. This includes the programmable switches matrix and logic blocks to support silicon reusability for implementing different logic circuitry with unlimited reconfiguration times. FPGAs generally offer substantial speedups at lower clock frequency than microprocessors, if the inherent parallelism in FPGA is fully utilized. The flexibility and promising performance of FPGAs enable them to bridge the gap between ASICs and microprocessors. FPGAs emerged as a result of advances in fabrication technology which enable fabrication of millions of transistors on a single silicon chip. Table 3.9 shows various Virtex FPGAs with their corresponding fabrication technology and numbers of logic cells/FPGA. The trend shows that, the number of logic cells increases beginning from early generation of Virtex FPGAs until the latest one (Virtex-7 FPGAs). This shows that more complex circuitry can be implemented in hardware as the size of a transistor to implement logic functions getting smaller.

Table 3.9: Virtex FPGAs and their corresponding fabrication technology

Year	**Xilinx Family**	**Technology (nm)**	**Logic cells**
1998	Virtex	180	27,648
1999	Virtex-E	130	73,008
2000	Virtex-EM	90	43,200
2000	Virtex-II	130	104,832
2002	Virtex-II Pro	130	125,136
2004	Virtex-4	90	200,448
2008	Virtex-5	65	331,776
2009	Virtex-6	40	758,784
2010	Virtex-7	28	1,139,200

3.6 The Alpha Data ADM-XRC-5LX

The Alpha Data ADM-XRC-5LX board as shown in Figure 3.11 is used in this research work for implementing the three reconfigurable sequence alignment architectures as outlined in Chapter 1.

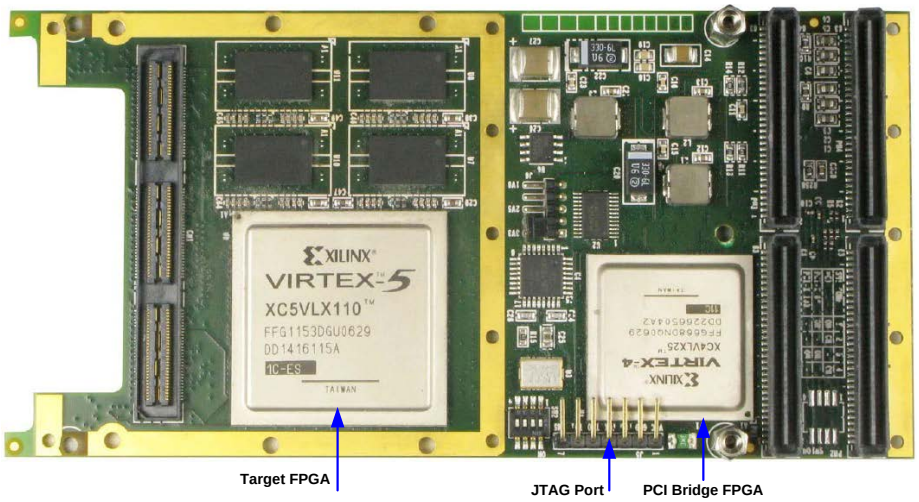


Figure 3.11: The ADM-XRC-5LX PCI mezzanine card [41]

This high performance PCI Mezzanine Card (PMC) comprises of a control FPGA which configures the PCI bus bridge interconnection. The PCI card physically conforms to the IEEE P1368-2001 Common Mezzanine Card standard and provides high performance PCI and DMA controllers with local bus speeds of up to 80MHz. It also provides additional memory of up to 1 GB from four different banks of 64x32 DDRII SDRAM, as illustrated in the functional diagram shown in Figure 3.12.

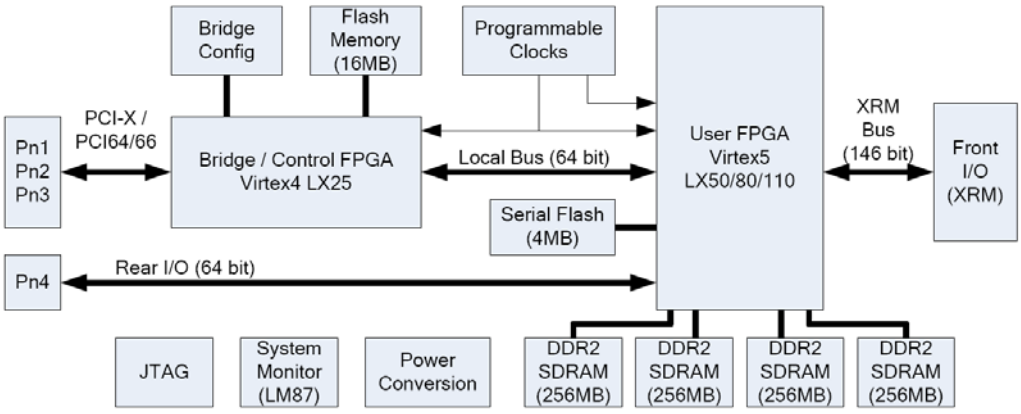


Figure 3.12: The ADM-XRC-5LX internal block diagram [41]

This card is also equipped with a user programmable clock, which can be set between 200-500MHz. The XC5VLX110-3FF1133 device on the Alpha Data board is the user FPGA, which is used in this work for prototyping the designed reconfigurable architectures. In general, the 65nm Virtex-5 FPGA comes in five different families; LX, LXT, SXT, TXT and FXT. These are used for different target applications. For instance, the user FPGA on the board is from the LX family, which is suitable for high-performance general logic applications. Other types of board such as those with SXT family are more applicable for high-performance signal processing applications. Details of the hardware resources available in the XC5VLX110 device are summarized in Table 3.10.

Table 3.10: The Xilinx XC5VLX110-3FF1153 hardware resources with the device maximum allowable operating frequency of 550MHz [42]

CLBs (Row x Col)	Slices(#)	LUTs (#)	FFs (#)	Max Distributed RAM (Kb)	Power PC	DSP48E slices	BRAM (36 or 18 Kb)
160 x 54	17,280	69,120	69,120	1,120	N/A	64	128 or 256

Figure 3.13 illustrates the overall system architecture of this work, which shows the Alpha Data ADM-XRC-5LX board and its interconnection with the host computer via the PCI-to-local-bus bridge. The board's internal structure follows the Mezzanine bus architecture, whereby the PCI bus is located between the host computer and the target FPGA.

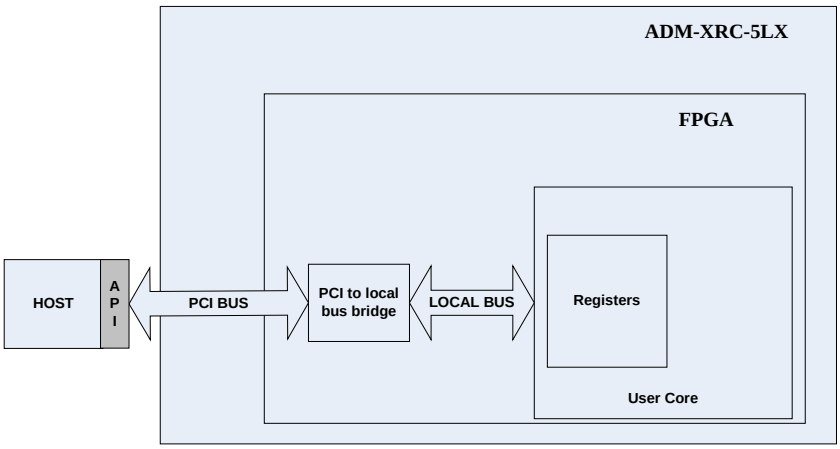


Figure 3.13: Top view of the ADM-XRC-5LX board and the host computer that is connected via the PCI local bus bridge controller

The control FPGA manages data transfer and the bus arbitration operations of the board. This Virtex-4 LX25 FPGA is configured by the Alpha Data proprietary firmware when the card is powered up, while the user core design is the proposed reconfigurable architectures implemented into the target FPGA. The communication registers facilitate both data transfer and communication between the host and the target FPGA. The PCI-to-local-bus bridge comprises one or more DMA engines to allow for rapid data transfer using the vendor-supported application programming interface (API) functions. The host CPU initiates the hardware using these API functions followed by configuring FPGA with the bitstreams before the subject sequences are transferred into the user FPGA to start the sequence alignment operation. A step-by-step flow of the aforementioned operations is given as follows;

1. Get the FPGA card and its memory space information and pass it to the host
2. Set the PCI local bus clock
3. Set the user core clock frequency
4. Configure FPGA with the bitstreams
5. Transfer database sequences using DMA to start sequence alignment
6. Close the FPGA card
7. Collect results back to the host

3.7 Summary and conclusions

FPGA is a general purpose integrated circuit which can be programmed by the end users. It is made up of programmable logic, routing interconnects and I/O interfaces. Most of the area of FPGA is covered by the programmable logic blocks. Xilinx has defined these as configurable logic blocks (CLBs). CLBs are reprogrammable and they can be used to prototype different applications. Today's FPGAs are essentially complex system-on-chip (SoC) architectures, which are incorporated with hard IPs such as hard microprocessor cores e.g. the Power PC 440 in the Virtex-5 FX family, embedded DSP slices, multi-gigabit transceivers and block RAMs. Hard IPs are permanently embedded in silicon die, while software IPs can be instantiated in the HDL design by the end users as required. Examples of soft IPs include the Pico Blaze and Micro Blaze soft core processors. Unlike the hard microprocessor core which is physically embedded in the FPGA, soft core processor peripherals are customizable based on the designer's requirements.

There are two distinct groups of Xilinx FPGAs; Spartan and Virtex FPGAs. Spartan FPGAs are low cost, high volume FPGAs, while Virtex FPGAs have been designed for high performance applications. The most primitive part of the CLB architecture is known as logic cell (LC) and among other elements, an LC comprises a LUT, a multiplexer and a register. Beginning from Virtex-5 family, logic cells are made up of the 6-input LUT compared to the 4-input LUT in their predecessors. With advancement in fabrication technology, the latest 28 nm Virtex-7 FPGAs offer up to two million LCs. In terms of area, power and performance efficiency, FPGA ranks second to the ASIC. Unlike the ASIC, the FPGA can be reconfigured after manufacture and has relatively faster times to market and lower production costs. This has enabled FPGAs to become more attractive, especially in the area of high performance computing. In terms of configuration methodology, modern FPGAs can be programmed either at compile time or at run time. The latter allows the efficient use of hardware resources in FPGA by time-multiplexing them to implement different applications on a silicon chip. Finally, the Alpha Data board with the Virtex-5 XC5VLX110-3FF1153 device as the target FPGA used in this research work was introduced. The board is based on the mezzanine bus architecture which is located between the host computer and the target FPGA. The card comes with proprietary firmware which is automatically loaded into the control FPGA when the card is powered up to initiate the overall system configuration.

Chapter 4
Design and FPGA Implementation of the Smith-Waterman Algorithm with Affine Gap Penalty

This chapter discusses the design and hardware implementation of the Smith-Waterman algorithm with the affine gap penalty. Prior to that, an introduction and background of the dynamic programming-based sequence alignment algorithm are presented. Then, previous hardware implementations of the optimal alignment algorithm are discussed. Following this, the double buffering technique adopted in the proposed core architecture is described. Discussion continues with an illustration and explanation of the proposed novel efficient scheduling hardware architecture for biological sequence alignment. Finally, the performance of the core is presented before conclusions are drawn and future work suggested.

4.1 Background

Dynamic programming (DP) was formalized by a mathematician, Richard Bellman, in the 1950s [43]. This is a powerful technique for solving a complex problem by breaking it down into smaller sub-problems and then solving each of these recursively to give an optimal solution. Dynamic programming is much like a ‘divide-and-conquer’ technique, except that it allows for the overlapping of sub-problems. It is used in various applications, including in biological sequence alignment. Sequence alignment is essentially a homology search comparing a newly discovered sequence against known sequences in a database, with the aim to infer clues about an unknown sequence. In this thesis, a newly discovered sequence is referred to as the query sequence and the known sequence is called the subject sequence. Examples of dynamic programming-based algorithms for performing sequence homology searches are the Smith-Waterman [14] and Needleman-Wusnch [13] algorithms. The former searches for the best local alignment between the two query and subject sequences, while the latter tries to align

entire regions of the two sequences during the search. The DP-based Smith-Waterman algorithm with linear gap penalty is shown in equation 4.1.

$$F(i,j) = \max\begin{cases} 0 \\ F(i-1,j-1)+s(x_i,y_j) \\ I_x(i-1,j-1)+s(x_i,y_j) \\ I_y(i-1,j-1)+s(x_i,y_j) \end{cases} \qquad (\ 4.1\ )$$

The local alignment algorithm was introduced in 1981 by T. F. Smith and M. S. Waterman [14]. Given a query sequence, $x = x_1, x_2, x_3..x_i...x_M$ (of length M) and a subject sequence $y = y_1,y_2,y_3..y_j...y_N$ (of length N), this recursive algorithm searches for the best alignment between sub-sequences of x and y. Scores for x_i and y_j are dictated by the largest score among the four alternatives. The $s(x_i,y_j)$ expression is a probabilistic score between amino acids x_i and y_j in sequences x and y respectively. This score is available in a two-dimensional matrix, called the substitution matrix. Figure 4.1 shows an example of this matrix, i.e. the BLOSUM50.

	A	C	D	E	F	G	H	I	K	L	M	N	P	Q	R	S	T	V	W	Y
A	**5**	-1	-2	-1	-3	0	-2	-1	-1	-2	-1	-1	-1	-1	-2	1	0	0	-3	-2
C	-1	**13**	-4	-3	-2	-3	-3	-2	-3	-2	-2	-2	-4	-3	-4	-1	-1	-1	-5	-3
D	-2	-4	**8**	2	-5	-1	-1	-4	-1	-4	-4	2	-1	0	-2	0	-1	-4	-5	-3
E	-1	-3	2	**6**	-3	-3	0	-4	1	-3	-2	0	-1	2	0	-1	-1	-3	-3	-2
F	-3	-2	-5	-3	**8**	-4	-1	0	-4	1	0	-4	-4	-4	-3	-3	-2	-1	1	4
G	0	-3	-1	-3	-4	**8**	-2	-4	-2	-4	-3	0	-2	-2	-3	0	-2	-4	-3	-3
H	-2	-3	-1	0	-1	-2	**10**	-4	0	-3	-1	1	-2	1	0	-1	-2	-4	-3	2
I	-1	-2	-4	-4	0	-4	-4	**5**	-3	2	2	-3	-3	-3	-4	-3	-1	4	-3	-1
K	-1	-3	-1	1	-4	-2	0	-3	**6**	-3	-2	0	-1	2	3	0	-1	-3	-3	-2
L	-2	-2	-4	-3	1	-4	-3	2	-3	**5**	3	-4	-4	-2	-3	-3	-1	1	-2	-1
M	-1	-2	-4	-2	0	-3	-1	2	-2	3	**7**	-2	-3	0	-2	-2	-1	1	-1	0
N	-1	-2	2	0	-4	0	1	-3	0	-4	-2	**7**	-2	0	-1	1	0	-3	-4	-2
P	-1	-4	-1	-1	-4	-2	-2	-3	-1	-4	-3	-2	**10**	-1	-3	-1	-1	-3	-4	-3
Q	-1	-3	0	2	-4	-2	1	-3	2	-2	0	0	-1	**7**	1	0	-1	-3	-1	-1
R	-2	-4	-2	0	-3	-3	0	-4	3	-3	-2	-1	-3	1	**7**	-1	-1	-3	-3	-1
S	1	-1	0	-1	-3	0	-1	-3	0	-3	-2	1	-1	0	-1	**5**	2	-2	-4	-2
T	0	-1	-1	-1	-2	-2	-2	-1	-1	-1	-1	0	-1	-1	-1	2	**5**	0	-3	-2
V	0	-1	-4	-3	-1	-4	-4	4	-3	1	1	-3	-3	-3	-3	-2	0	**5**	-3	-1
W	-3	-5	-5	-3	1	-3	-3	-3	-3	-2	-1	-4	-4	-1	-3	-4	-3	-3	**15**	2
Y	-2	-3	-3	-2	4	-3	2	-1	-2	-1	0	-2	-3	-1	-1	-2	-2	-1	2	**8**

Figure 4.1: The BLOSUM50 substitution matrix (re-arranged into alphabetical order) with 20 by 20 elements of amino acid residues[3].

This substitution matrix represents biological relationship of amino acids x_i and y_j. in the form of integer scores, as shown in Figure 4.1. Note that, the entries on the main diagonal as highlighted in bold represent identical residue pairs of the amino acids.

4.1.1 Alignment matrix computation

An alignment matrix represents the degree of similarity between each residue pair of sequences *x* and sequence *y*. A score associated with each pair is calculated using the *F(i,j)* matrix as discussed in section 4.1. The dynamic programming algorithm calculates the *F(i,j)* matrix and builds it recursively from the first segment $x_{1\ldots i}$ of *x* up to x_i and the first segment $y_{1\ldots j}$ up to y_j, resulting in an alignment matrix of size *M* x *N*. To explain the recursive operations in an alignment matrix, the global alignment algorithm in equation 4.2 as discussed in chapter 2 is used as a reference.

$$F(i,j) = \max\begin{cases} F(i-1,j-1)+s(x_i,y_j) \\ I_x(i-1,j-1)+s(x_i,y_j) \\ I_y(i-1,j-1)+s(x_i,y_j) \end{cases} \qquad (\ 4.2\ )$$

The alignment matrix which is used to calculate the alignment scores between sequences *x* and *y* is shown in Figure 4.2.

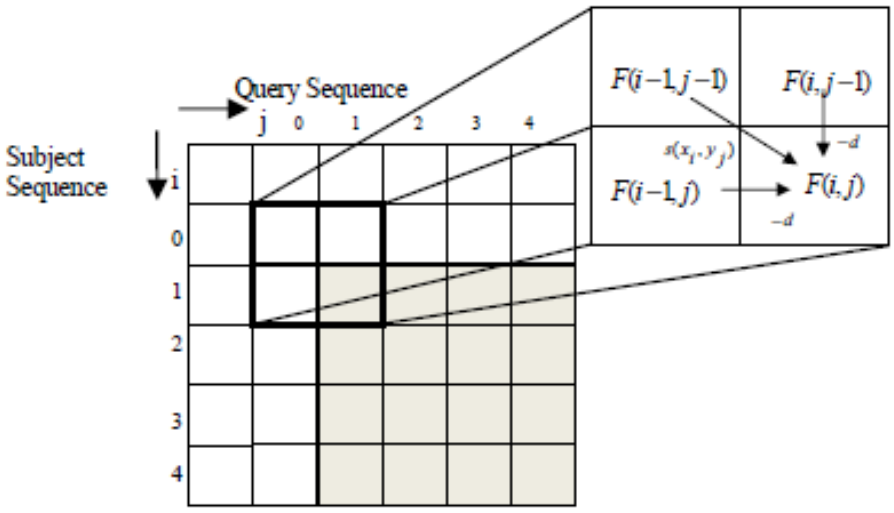


Figure 4.2: Computing *F(i,j)* in an alignment matrix of size (*M* x *N*)

In the alignment matrix, the alignment score *F(i,j)* is indexed by *i* and *j*, where *i* is an index for each subject sequence residue and *j* is an index for each query sequence residue. The three adjacent elements i.e. the diagonal element *F(i-1,j-1)*, the top element *F(i,j-1)* and the left element *F(i-1,j)* are the data dependency of the *F(i,j)*, whereas the *F(i,j)* score is the highest score from any of these three possible alternatives. In the case of local alignment, the *F(i,j)* score considers another value i.e. zero in the maximum expression as discussed in section 4.1. $s(x_i,y_j)$ is the probability score of residues x_i and y_j. Before constructing the alignment matrix, boundary values are also required for the alignment matrix. The *F(0,0)* is set to zero as it obviously represents no alignment in either sequence *x* or *y*. It is thus always set to zero in both local and global alignment. In the case of global alignment, the *F(i,0)* cells, which represent the alignment of prefix *x* to all gaps in *y*, must be set to *-id*. Similarly the *F(0,j)* is set to *–jd* as it represents the alignment of prefix *y* to all gaps in the *x* direction. In the case of local alignment, all boundaries i.e. *F(i, 0)* and *F(0, j)* are set to zero.

4.1.2 Aligning sequences with optimal results

In the linear gap penalty as discussed in section 2.3.2, the constant gap penalty *d* penalizes gaps of length *g* linearly, i.e., *penalty (g) =-gd*. Then, a more efficient gap penalty model was introduced by Gotoh in 1982 [15]. The global alignment with the affine gap penalty model is shown in equations 4.3, 4.4 and 4.5.

$$F(i,j)=\max\begin{cases}F(i-1,j-1)+s(x_i,y_j)\\ I_x(i-1,j-1)+s(x_i,y_j)\\ I_y(i-1,j-1)+s(x_i,y_j)\end{cases} \qquad (4.3)$$

$$I_x(i,j)=\max\begin{cases}F(i-1,j)-d\\ I_x(i-1,j)-e\end{cases} \qquad (4.4)$$

$$I_y(i,j)=\max\begin{cases}F(i,j-1)-d\\ I_y(i,j-1)-e\end{cases} \qquad (4.5)$$

In this gap penalty model, a constant gap cost is given when opening a new gap (gap opening or *d*), while a linear and often smaller gap penalty is given for subsequent gap extensions (*e*), i.e., *penalty(g)=-d-(g-1) e*. *F(i,j)* is the best score up to *(i,j)* where residue x_i is aligned to residue y_j. $I_x(i,j)$ is the best score where residue x_i is aligned to a gap, and finally the $I_y(i,j)$ is the best score where residue y_i is aligned to a gap. This type of algorithm is more complex than those using the linear gap penalty. In terms of accuracy, the affine gap function models the gap penalty in such a way so as to be closer to the biological phenomenon compared to alignment with the linear function, and thus this algorithm produces more accurate results and is widely-used in sequence alignment algorithms. In the case of local alignment, zero is added to the maximum expression of *F(i,j),* and the alignment algorithm with affine gap penalty is referred to as the Gotoh algorithm [44].

4.1.3 Systolic array

Performing sequence alignment using DP-based algorithms is time consuming due to their quadratic time complexity. Alternatively, these algorithms are accelerated in hardware using a linear systolic array. The systolic array was introduced by H.T.Kung in 1978 [45], and is widely-used to accelerate all DP-based algorithms in hardware. In sequence alignment, the one-dimensional systolic array is used to accelerate the $O(n^2)$ algorithm, which results in linear time complexity as compared to the sequential implementation of the algorithm in a standard microprocessor. A linear systolic array is an arrangement of processing elements (PEs) in a one dimensional array, as shown in Figure 4.3. Each PE is connected to its neighboring PEs and data flows synchronously across the array between the adjacent PEs. For instance, aligning sequence *x* of length *M* and sequence *y* of length *N* using the systolic array results in computation time complexity of *O(M+N-1)*. This is due to the anti-diagonal computation flow of the systolic array as illustrated in Figure 4.3. In hardware, the PE systolic array holds one query residue at a time in order to calculate the alignment matrix score *F(i,j)* of residue x_i and y_j. The PE computes the elementary operations of the alignment algorithm as each subject sequence's residue is shifted at each processing step through the array of PEs.

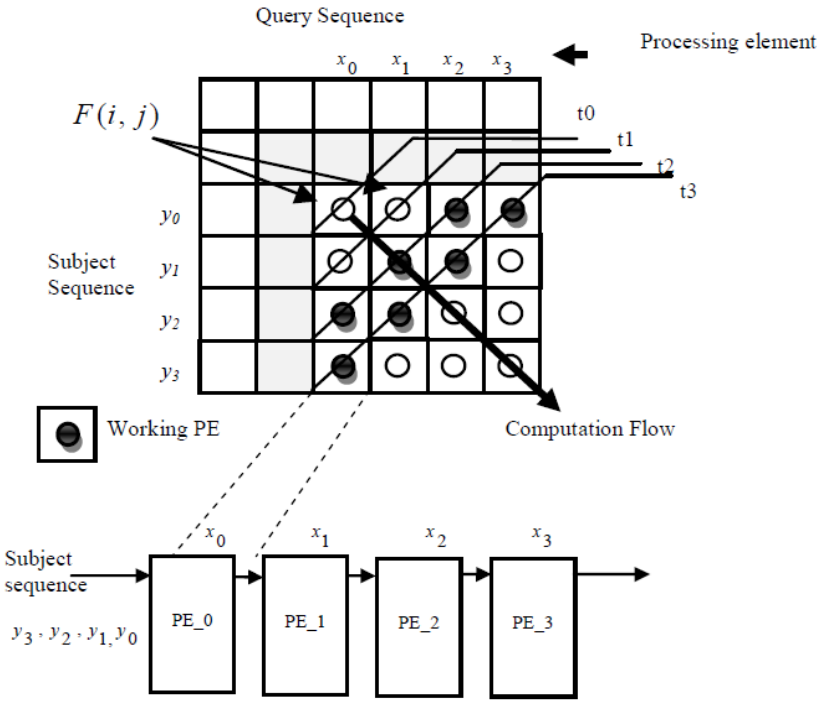


Figure 4.3: Alignment matrix computation using a linear systolic array

4.2 Prior work on FPGA-based dynamic programming for sequence alignment

The acceleration of string pattern matching problems in hardware has started as early as 1980 [46] with the special purpose VLSI chip used by Foster and Kung to compute an algorithm for the string pattern matching problem using systolic array architecture. The term systolic array was coined by Kung and Leiserson in 1978 at the Carnegie-Mellon University [47]. This one-dimensional array enables the acceleration of DP algorithms by means of computing the recursive equation in anti-diagonal flow instead of sequential flow as in a standard microprocessor. Another early study which implemented the DP algorithm on special-purpose VLSI chip was reported by Lipton and Lopresti in 1985. The sequence edit distance algorithm was implemented in the processing element and a total of 30 systolic processors were used for the acceleration of the pattern matching problem. Following that, the Princeton Nucleic Acid Comparator (P-NAC) was reported by Lopresti in 1987 [48]. This VLSI core performed DNA sequence comparisons and achieved speeds 125 times faster than a minicomputer (DEC VAX 11/785). In the early

1990s, Field Programmable Gate Arrays (FPGAs) were used to accelerate the algorithm using a linear systolic array. SPLASH was among the first off-the-shelf FPGA-based sequence edit distance accelerators, and was reported by Hoang and Lopresti [49]. It comprised of 24 PEs where each implemented the sequence edit distance algorithm. However, at that time FPGAs were not as competitive as they are today. Thus, other parallel architectures were developed, including the single instruction multiple data (SIMD) architectures such as micro grain array processor (MGAP) [50] in 1994, Kestrel [51] in 1996 and Fuzion [52] in 2002. These parallel architectures offer considerably higher speed-up performance than a standard desktop solution at the expense of higher design and programming costs. Then, the special-purpose biological sequence alignment accelerators such as the Biological Information Signal Processor (BISP) [53] in 1991, Systolic Accelerator for Molecular Biological Applications Biological Sequence Comparative Analysis Node (BioSCAN) [54] in 1996 and Systolic Accelerator for Molecular Biological Applications (SAMBA) [55] in 1997 were developed. Due to the non re-programmable nature of these special-purpose architectures, different needs for the implemented algorithm could not be tuned both at compile time and at run-time.

Over the last decade, advances in CMOS process technology have enabled the fabrication of millions of transistors onto a single silicon chip. This allows for the acceleration of more complex functions in FPGAs. In addition, their capability of implementing parallel processing, as offered by special-purpose architectures with the added convenience of re-programmability, has led FPGAs to become an attractive platform for hardware acceleration. This has led to the implementation of the Smith Waterman and Needleman-Wunsch algorithms with the affine gap penalty on FPGAs. However, since the completion of the Human Genome Project (HGP) in 2003, the numbers of biological sequences in databases have increased exponentially [56]. Biological sequences are often hundreds if not thousands of residues in length and therefore considerable logic resources are required in order to process them, which is a challenge even with modern FPGAs. The implementation of the Smith-Waterman with affine gap penalty on FPGA as in [57] was among the early works reported in literature. It was reported in 2002, when Yamaguchi et al. implemented the Smith-Waterman with affine gap function on the RC 1000-PP Celoxica board with Virtex-II FPGA. During that time, Virtex-II was the latest FPGAs and the Xilinx XCV2000E device fitted a maximum of 144 processing elements. The core with a clock frequency of 40 MHz

searched a query sequence of 2048 residues in length against a database of 64 million sequences in 34 seconds, representing a speed about 330 times faster than the same sequence alignment executed on an Intel Pentium III desktop computer with an operating frequency of 1 GHz. In 2005, Oliver et al. used run time reconfiguration (RTR) by reconfiguring PEs to enable hardware re-use for cases of query sequences of lengths longer than the maximum PEs. This way, different circuits were configured on demand during alignment matrix computation. The algorithm was then implemented on the RC200 FPGA Mezzanine PCI-board with the Virtex-II FPGA [40]. With 33,792 slices and 144 of blocks RAM, the XC2V6000 successfully fitted 168 affine gap PEs with a clock frequency of 45 MHz. The core searched for various query sequence lengths against the Swiss-Prot protein database release 42.5, which contained 138,922 sequences or 51,161,444 amino acids. The core achieved speed-up of 125x that of an optimized C-program which ran on the Intel Pentium IV 1.6 GHz processor. However, although, the reported FPGA implementations re-used the silicon chip by using RTR to support longer query sequences, the reconfiguration bandwidth was limited and the extra logic resources needed for PE reconfiguration were still considerable at for example 80 ms to reconfigure the XC2V6000 device.

Other reported FPGA implementations include the ones presented by Jacobi et al. [58] and VanCourt and Herbordt [59]. These works focused on the Smith–Waterman with linear gap penalty, and in these two architectures sequence alignment with accelerated trace-back method was proposed. However, this technique required more memory allocation for the trace back pointers (diagonal, right and down arrows) which could be implemented in software. Neither the reported cores implemented RTR and therefore the search query sequences were limited to the number of PEs, as in the one reported by Hoang [49]. Alternatively, the PE systolic arrays were scaled on multiple FPGAs, as reported by Abouellail et al. [60] in 2007 in order to enable sequence alignment with longer query sequence. However, this led to cost ineffectiveness and not promoting hardware re-use. Another approach has also been reported in [61], [62], [63] and [64]. In this approach, longer query sequences are compared against a database sequence by partitioning the S-W algorithm into smaller alignment steps and processing them sequentially in multiple passes over the same systolic array, in the so-called folding technique. The folded systolic array architecture required a feedback FIFO, as shown in

Figure 4.4, to temporarily hold subject sequences and other intermediate data for processing and subsequent pass computation.

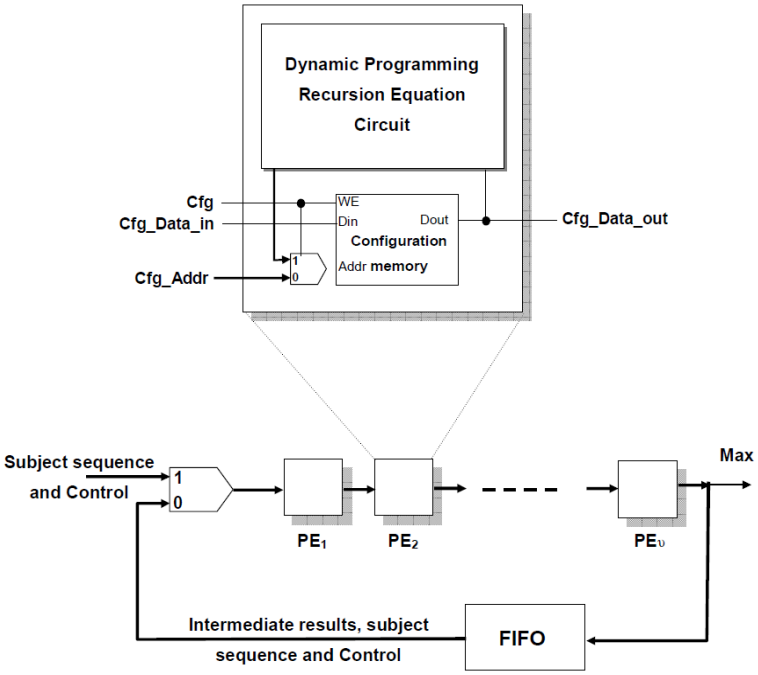


Figure 4.4: The configuration memory to update the PE with different substitution matrix columns through the serial configuration chain in the PE with fixed size systolic array [61]

This technique has enabled hardware re-use rather than replicating PE systolic arrays in multiple FPGAs. Among other challenges, processing over multiple passes requires a different set of substitution matrix columns for each pass computation. The substitution matrix column (henceforth referred to as the configuration element (CE)) is dictated by the query residue held by the PE for an alignment matrix computation. Examples of such an approach were reported by Jiang et al. [65] in 2007 and Benkrid et al. [61] in 2009. This technique enabled the PE to be reused at the expense of extra time for CE configuration since each configuration element required different coefficients for each pass computation. The coefficients which were stored in the PE were updated through a serial configuration chain before subsequent pass computation commenced.

In 2007, Zhang et al. [66] from the Altera Corporation introduced a new storage method to reduce block RAM usage in the PE. In this approach, entire substitution matrix coefficients were pre-stored in the PE and an address encoding method was used

to access individual coefficients in the matrix for computation. This enabled rapid access to the substitution matrix as it was stored in the PE. In 2011, Yamaguchi et al. [67] used almost the same method to store a substitution matrix in the PE for multiple-pass computation. Although this technique took significantly less configuration time compared to the use of a serial configuration chain, the restricted memory resources limited the scalability of the PE and different encoding schemes may be required for different types of substitution matrix.

Taking into consideration the aforementioned hardware implementation issues, a novel sequence alignment core architecture with fixed CEs is proposed. To manage the fixed CEs for alignment matrix computation and CE configuration in a folded systolic array, an efficient scheduling strategy based on the double-buffering technique is adopted in the core. In this architecture, the scheduling strategy is referred to as overlapped computation and configuration (OCC). Prior to that, the following section first discusses the novel PE architecture with multiple CEs (*n*CEs) which is published in 2011 and extensively reported by Isa et al. in [64].

4.3 The PE with multiple configuration elements

This PE with multiple configuration elements (*n*CEs) was the initial sequence alignment core architecture proposed in this research work to overcome the issue of PE dependency on block RAMs in FPGA. Typical PE architecture as reported in literature utilized the restricted memory resources to store entire substitution matrix. Although this approach enables rapid access of the coefficients, only a column of the substitution matrix which is represented by a query residue held by the PE, is used during alignment matrix computation. Therefore, the PE with multiple configuration elements is proposed to hold only required columns of substitution matrix, where the numbers of CEs in the PE are based on the number of processing passes required. This is realized in hardware by first pre-loading only query-related substitution matrix columns into their corresponding CEs during the CE configuration phase. These columns are dictated by the query residues held by the PE in the folded systolic array architecture. This enables alignment matrix computation to start without the need to reconfigure PE with new coefficients. Figure 4.5 illustrates internal architecture of the proposed PE.

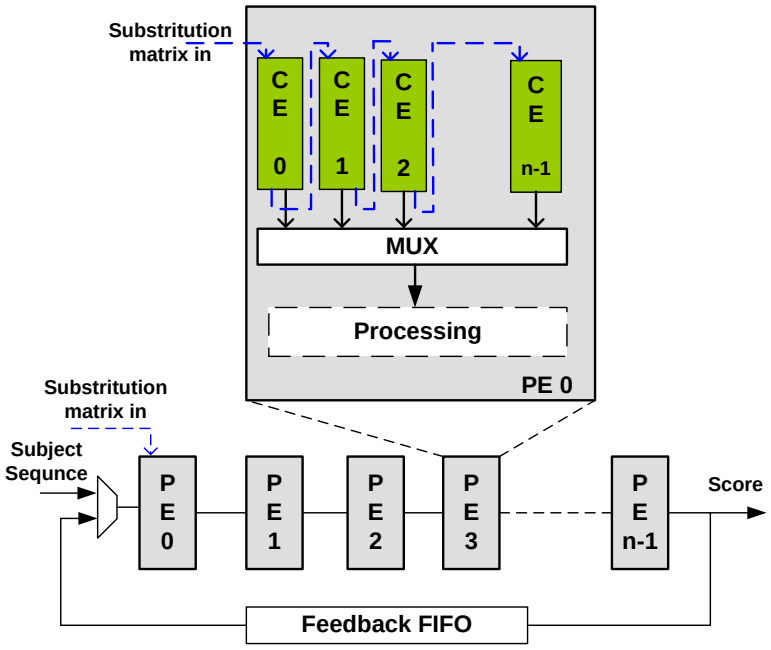


Figure 4.5: The proposed PE with multiple configuration elements (n_{CEs})

This way, each pre-stored column of substitution matrix in the CE is utilized in turn during alignment matrix computation based on their corresponding number of

computational passes. In terms of hardware resources utilization, these columns of coefficients are temporarily stored in the FPGA's look-up table rather than storing them in the restrictive memory resources. The novel PE architecture is designed to enable alignment matrix computation in folded systolic array without relying on the limited embedded memory resources. This multiple CEs architecture is suitable for the case of computing alignment scores of longer query sequences than the physically implementable number of PEs in an FPGA.

During alignment matrix computation, the PE is only computing alignment score of one query residue. Then, alignment score of subsequent query residue is calculated in turn in subsequent fold computation. Therefore computation in *n*-pass requires *n*CEs in the PE to allow for *n*-pass computation. This enables smooth computation between multiple-pass computations since all CEs are configured during initial configuration using the serial configuration chain as illustrated by the blue-dotted line in Figure 4.5. Although the PE architecture reduces design dependency with the restricted memory resources by utilizing the abundant CLB logic slices to store substitution matrix columns as well as enabling PE scalability, the replication of CEs in the PE incurred n_{CE} area overheads per PE. Additionally, the initial configuration chain required an *n*-fold increase in PE configuration time. This leads to a better PE architecture which optimizes logic resources in the PE by having a fixed number of CEs. In the new PE architecture, only two CEs are proposed, where each CE is used one after another during alignment matrix computation in a folded systolic array. In an effort to manage the fixed CE resources in *n*-pass computation, an efficient scheduling strategy based on the double buffering technique is adopted in the controller to efficiently manage the CEs one after another. A detailed explanation regarding the fixed CEs and it scheduling strategy is presented in the next section.

4.4 The efficient scheduling strategy

The proposed core is useful in processing long sequences where it is not possible to allocate sufficient PEs to the FPGA in hand. Rather than replicating PE-related substitution matrix coefficients at each pass (or fold), the proposed architecture uses a fixed number of configuration elements (equal to 2, as shown in Figure 4.6 (a), namely CE_0 and CE_1) regardless of the folding factor.

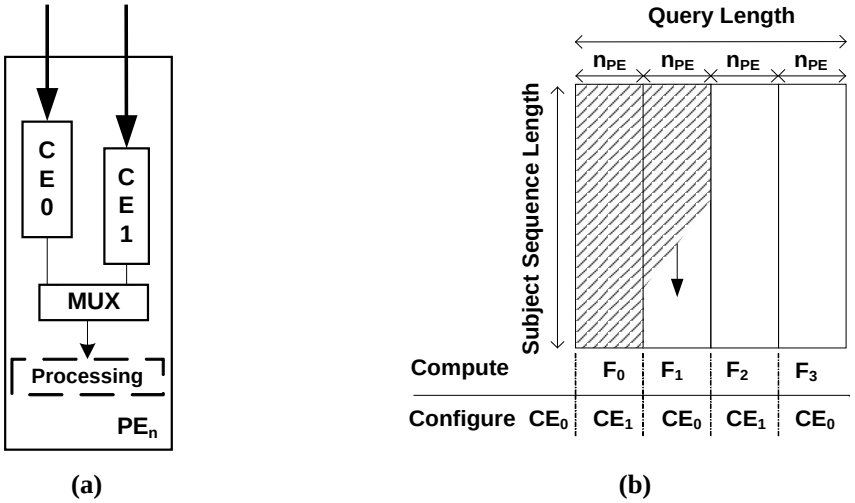


Figure 4.6 : (a) Internal PE structure with fixed configuration elements (CEs).
(b) Computation and configuration over the same systolic array.

Alignment matrix computation uses one CE (CE0) while configuring the content of another element (CE1) for the subsequent pass, and vice versa in the other subsequent pass. In the example shown in Figure 4.6 (b), a folding factor of four is assumed, where a query sequence of length $4\ n_{PE}$ is to be aligned. However, only n_{PE} can be implemented in hardware. The passes or folds required are denoted as F_0, F_1, F_2 and F_3 in Figure 4.6 (b). To allow efficient scheduling between configuration and computation, all CE_0 elements in the PEs are updated during the *Initial Config.* phase as depicted in Figure 4.7 (a). As CE_0 is ready, the first computation starts (F_0) and during this time CE_1 of all PEs is updated with new coefficients for the next fold computation (F_1) (labeled as Overlap 1). F_1 starts computation at t_4 when the tail of the current subject sequence leaves the first PE. During F_1 alignment computations, F_0 finishes its task once the tail of the current subject sequence (see Figure 4.7 (b)) leaves the last PE, and the second overlap operation (Overlap 2) then occurs where CE_0 will be updated with new coefficients for subsequent fold computation (F_2).

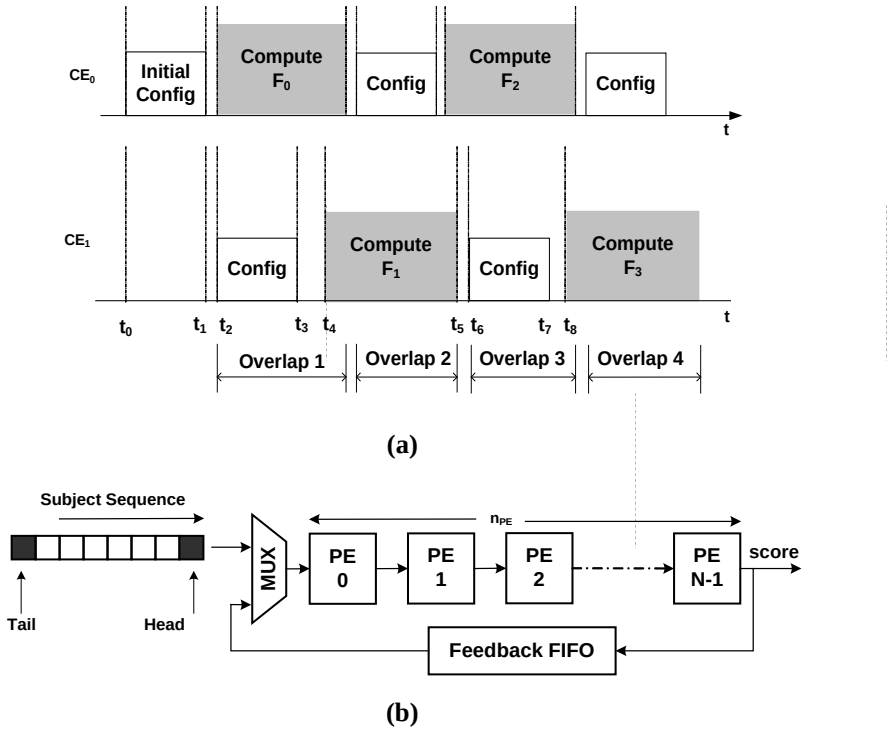


Figure 4.7: (a) The efficient scheduling strategy between CE configuration and alignment matrix computation. (b) Subject sequence flows through processing elements of size n_{PE} in folded systolic array architecture.

This overlapping operation continues until the last fold (Overlap 4 as in Figure 4.7 (a)). Note that, during each last computation fold, CE_0 is configured with new coefficients for the new subject sequence in the database. This cycle continues until all subject sequences in the database are exhausted. In this research work, this approach is referred to as overlapped computation and configuration (OCC).

4.5 The novel hardware architecture

This section discusses the novel architectures for the sequence alignment core which includes the PE, the QUERY LOADER, the PARALLEL LOADER and the OCC Scheduler (MAIN CONTROLLER). These logic units are designed to implement an efficient scheduling technique, as mentioned in section 4.3. The overall system architecture of the biological sequence alignment core with the proposed architectures

are highlighted and illustrated in Figure 4.8. Note that all computational parameters, including the data width (*dw*) and compute data width (*cdw*) are parameterizable.

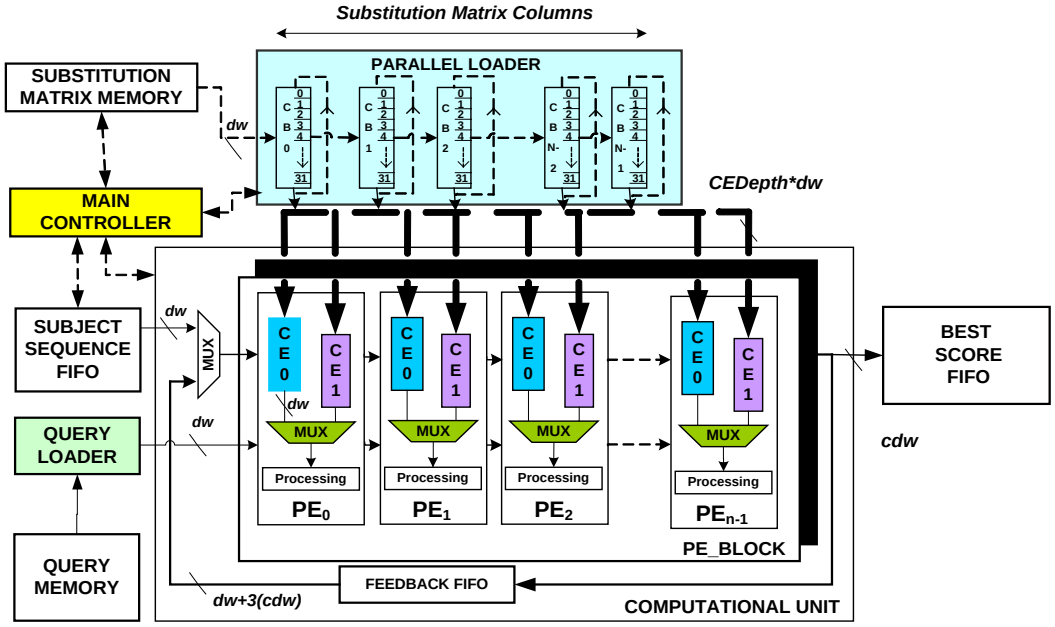


Figure 4.8: The overall core architecture with the double buffering CEs

The PE computes elementary functions of the DP algorithm and communicates with the next PE using regular interconnections to form a linear systolic array of PEs (PE_BLOCK). Unlike typical folded Smith-Waterman architectures, the proposed PE has only two CEs and thus proper scheduling is required to alternately use CE_0 and CE_1 for configuration and alignment matrix computation. This way, a CE is configured with different probability score tables at different folds while another CE holds a column of substitution matrix scores for the corresponding fold computation. This enables the efficient use of logic resources during multiple-pass computations. Moreover, with the efficient scheduling strategy, the overall system throughput increases significantly.

To implement the overlapped computation and configuration strategy in the sequence alignment core, the MAIN CONTROLLER schedules both configuration and the computation modes to run simultaneously. This operation virtually removes the time taken for CE configuration during every fold computation. Another logic unit which is crucial for efficient scheduling between tasks in computing the alignment matrix and configuring the CE for subsequent pass computation is the PARALLEL LOADER. As its name implies, this loader is designed to configure CEs in parallel with bounded CE

configuration time regardless of the number of PEs or the length of the query sequence. This way, the time taken to configure the CE in all PEs is less than the time which has elapsed in computing the alignment matrix. This enables the smooth scheduling of the concurrent operations (alignment matrix computation and CE configuration) during each fold computation. Details regarding the three novel architectures are given in the following sub-sections.

4.5.1 The query loader

The query loader is responsible for mapping the CE with its corresponding query sequence residue. The query residue-to-CE mapping operation is a part of the CE configuration task which is required prior to the preloading of the substitution matrix coefficients as outlined in section 4.3. To illustrate the operation of the Query Loader, a query sequence of a length of ten residues is assumed as an example and only five processing elements can be implemented in hardware, as shown in Figure 4.9 (a). Therefore, the query sequence is partitioned into two (in the case of a fold of two) as illustrated in Figure 4.9 (b), and the computation of the query sequence proceeds in two passes, F_0 and F_1 by re-using the folded PE systolic arrays. The query loader initially maps CE_0 elements in the PE_0 up to PE_4 with the first five residues. The remaining residues in the query sequence will be read into CE_1 during the computation of F_0. The time required (in clock cycles) to map the query sequence residue is given in equation 4.6.

$$t_{query} = \frac{Q_{length}}{k} = n_{PE} \qquad (4.6)$$

where, Q_{length} is the length of the query sequence and k is the number of folds required. The number of fold is calculated by dividing Q_{length} by n_{PE}.

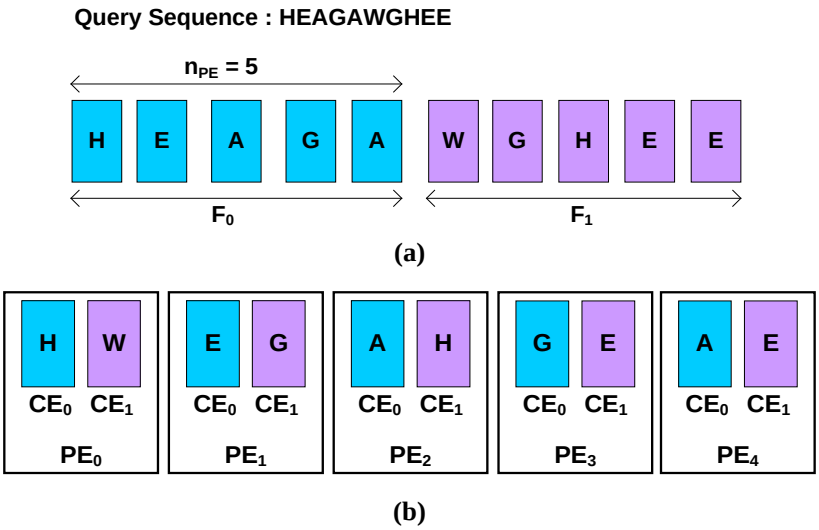


Figure 4.9: (a) Query sequence residues partitioned into two portions, the first to compute during F_0 and the second during F_1. (b) The corresponding residue-to-CE mapping in hardware.

For the sake of simplicity, this example shows CE mapping in the case of a fold of two. For higher fold factors, the CE mapping task follows the same procedure, where all CE_0 elements are allocated for odd-numbered fold computations (F_1,F_3,F_5 and so on), whereas all CE_1 elements are allocated for even-numbered fold computations (F_0,F_2,F_4 and so on). Details of the loading mechanism are elaborated in the following section.

4.5.2 The proposed parallel loader

The main function of the parallel loader is to simultaneously load all of the PEs of a pairwise sequence alignment array with their corresponding substitution matrix columns. This allows for efficient data transfer since the configuration time is significantly reduced to $1/k \times n_{PE}$ as compared to that in conventional serial configuration techniques, where k is the fold factor and n_{PE} is the number of PEs. Consequently, a PE with only two configuration elements could be used for any folding factor, so that one CE is used for alignment matrix computation, and the other CE is updated with a column of substitution matrix scores for the subsequent fold computation. The decision about which column to load into the CE is dictated by the QUERY LOADER, as outlined in section 4.5.1. Figure 4.10 illustrates the internal elements of the PARALEL LOADER.

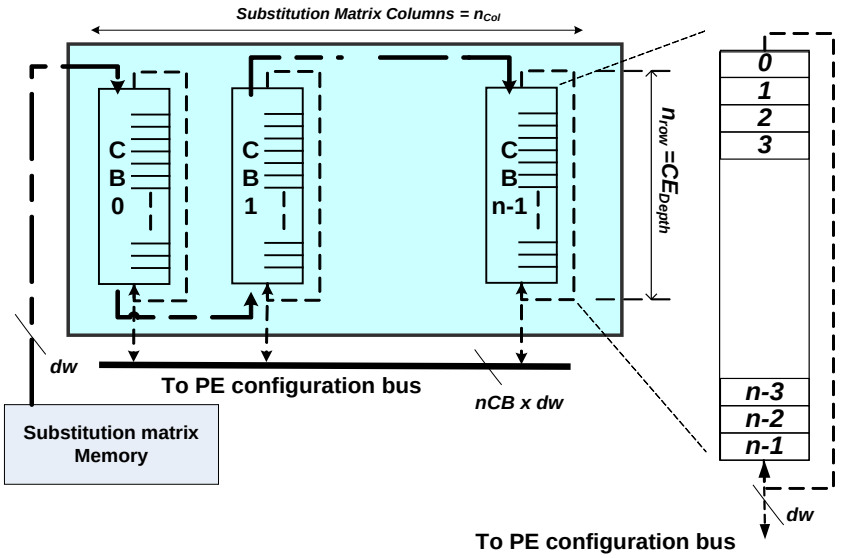


Figure 4.10: The circular buffers in the parallel loader. Each circular buffer holds a column of substitution matrix scores

Both the n_{row} and the n_{col} of the PARALLEL LOADER are parameterizable, and for this implementation both are set to 32 elements. The loader is made up of n_{CB} circular buffers, which are implemented efficiently using shift registers based on the FPGA's Look-up Tables (LUTs), also referred to as the SRL32 [68]. The loader with n_{CB} circular buffers holds the columns of the substitution matrix e.g. $n_{CB} = n_{col} = 32$. The buffer has n_{row} shift registers, with each shifting one element of a particular substitution matrix row into the buffer, in turn, every clock cycle. The word length of the substitution matrix elements, *dw*, is parameterizable. In the case of the BLOSUM 50, 5-bit two's complement is enough to represent its elements, in which case the loader operates as a 5-bit serial-in-serial-out shift register during initial configuration mode, and once in running mode, it operates as a 5-bit serial in n_{col} *x dw* -bit parallel out circular shift register.

4.5.2.1 Initial configuration mode

The right shift operation is the fundamental operation of the loader during the initial configuration mode. The operation starts by serially shifting elements of a given substitution matrix column by column into the corresponding buffers which are

pipelined together in a long chain. Each element is shifted into the buffer chain every clock cycle. Consequently, all of the substitution matrix elements of one column are completely loaded into the buffer chain within n_{row} clock cycles. Note that the thick broken line arrow in Figure 4.10 depicts the flow of the shift operation during the configuration mode. It begins to fill the last buffer, i.e. CB_{n-1}, with the first n_{row} elements in the last column of the substitution matrix and continues with the following n_{row} elements to buffer $CB_{n-2.}$ This sequential shift operation continues until CB_0. This way, all scores will be loaded into the buffer chain according to their corresponding column. Once all of the scores are completely loaded, so that the substitution matrix memory read is finished, the loader is ready to configure the PEs. The initial configuration time (in clock cycles) to read an entire substitution matrix into the loader depends on the size of the substitution matrix, and is mathematically expressed in equation 4.7.

$$t_{initialload} = n_{col} \times n_{row} \qquad (4.7)$$

where, n_{col} is the number of columns and n_{row} is the number of rows in the substitution matrix.

4.5.2.2 Running Mode

During this mode, all elements in a buffer are circulated every clock cycle following the direction of the arrow with the thin dotted line as shown in Figure 4.10. Data circulation within the circular buffers ensures that valid scores are available for PE configuration within a maximum duration of $2n_{row}$, as expressed in equation 4.8. This means that the worst case configuration time for all CEs is $2n_{row}$ clock cycles.

$$tCE_{config} \quad \leq \quad 2 \times n_{row} \qquad (4.8)$$

In the timing diagram shown in Figure 4.11, the substitution matrix memory is assumed to be already filled with probability scores. During the initial configuration mode, the loader operation is marked by the BUSY signal being HIGH. Once all scores are fully loaded, the loader is ready for PE configuration, and thus synchronization pulses (SYNCH_PULSE) are emitted every n_{row} clock cycles, whereby at each pulse interval valid scores are available on the PE configuration bus for CE configuration. The CE configuration happens at any stage during this interval, when its own probability

scores are output by the circular buffers. Indeed, the query residue inside each CE selects its corresponding substitution matrix column.

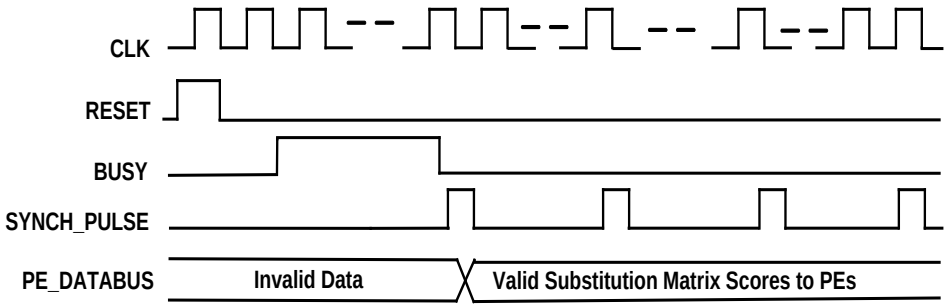


Figure 4.11: Valid substitution matrix scores available to the PE during SYNCH_PULSE intervals

4.5.3 Internal PE Architecture

In this work, the PE implements the Gotoh algorithm [15] and the pseudo code of the algorithm is illustrated in Figure 4.12. This algorithm is essentially a more accurate version of the local alignment algorithm, whereby the affine gap penalty model as proposed by the Gotoh [15] is adopted into the original Smith-Waterman algorithm.

Input

1. Query Sequence x, length : *M residues*
2. Subject Sequence y, length : *N residues*
3. Substitution Matrix Coefficient : $s(x_i,y_j)$
4. Gap Open Penalty : d
5. Gap Extension Penalty : e

Initialization

for $i \leftarrow 0 \; to \; M$
$M(i,0) \leftarrow 0$
$I_x(i,0) \leftarrow -\infty$
$I_y(i,0) \leftarrow -\infty$
for $j \leftarrow 0 \; to \; N$
$M(0,j) \leftarrow 0$
$I_x(0,j) \leftarrow -\infty$
$I_y(0,j) \leftarrow -\infty$

Recursion

for $i \leftarrow 0 \; to \; M$
for $j \leftarrow 0 \; to \; N$

$$F(i,j) \leftarrow \max \begin{cases} 0, \\ F(i-1,j-1)+s(x_i,y_j) \\ I_x(i-1.j-1)+s(x_i,y_j) \\ I_y(i-1.j-1)+s(x_i,y_j) \end{cases}$$

$$I_x(i,j) \leftarrow \max \begin{cases} F(i-1,j)-d \\ I_x(i-1,j)-e \end{cases}$$

$$I_y(i,j) \leftarrow \max \begin{cases} F(i,j-1)-d \\ I_y(i,j-1)-e \end{cases}$$

Figure 4.12: The pseudo code of the Gotoh local alignment algorithm [44]

The inner structure of the PE which implements the Gotoh local alignment algorithm is illustrated in Figure 4.13. The PE is designed so that all computational parameters, including the gap data width (*gdw*) and the depth of the CE (CE_{Depth}) are parameterizable. In this architecture, the *gdw* is four bits, which is enough to represent the gap open and gap extension penalty scores for the affine gap function. On the other hand, the CE_{Depth} is set to 32 elements (5 bits), which suffices for DNA (with nucleotides of A, G, T and C) and protein (with 20 amino acids) sequences. The main task of the PE is to calculate the elementary operations of the local alignment algorithm. The affine gap penalty PE consists of three arithmetic units; the best score ($F(i, j)$) of residues x_i and y_j, the best score of insertion with respect to the x direction ($I_x(i, j)$) of residue x_i and y_j, and the best score with respect to the y direction ($I_y(i, j)$) of residues x_i and y_j. All of these units are illustrated in Figure 4.13 from the top down to the bottom of the shaded boxes respectively.

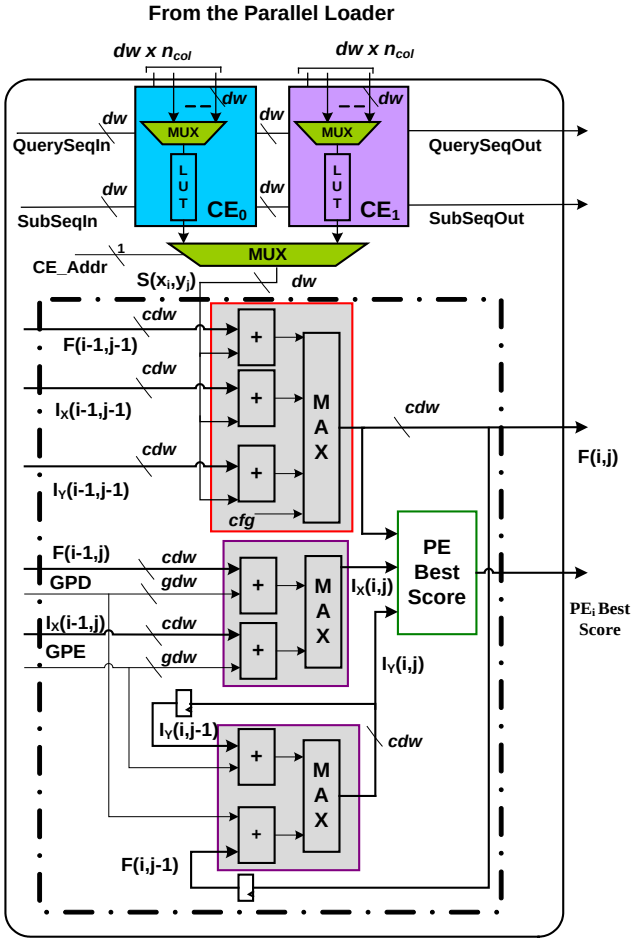


Figure 4.13: Internal PE architecture for the Gotoh algorithm

The *Cfg* input is added to the maximum expression to tackle different types of alignment such as local and global alignments. In this architecture, the *Cfg* flag is set to '0' as it implements the local alignment algorithm. This allows the saturation of alignment scores to zero. In the case of global alignment, the *Cfg* input is set to'1'. The PE Best Score unit calculates the 'maximum so far' of the alignment scores, taking into account the PE_i and PE_{i-1} best scores. Then it propagates the score to PE_{i+1} in a chain across the PE systolic arrays. If the accumulated score satisfies a given threshold value, the best score of the last PE with its corresponding subject sequence address are stored in the Best Score FIFO; otherwise, the score and the subject sequence are disregarded. As mentioned in section 4.3, both configuration and computation modes run simultaneously except during the *Initial Config.* mode. For the sake of clarity, each mode is explained separately in this section.

During the configuration mode, the query sequence residue fetches its corresponding substitution matrix column via the CE_Addr port. The CE temporarily holds a column of substitution matrix scores for alignment matrix computation. Since both CEs are used alternately for computation, a multiplexer is used to fetch probability scores either from CE_0 or CE_1, whereby the selection is determined by the CE_Addr port. The CE selection strategy is based on the computational passes. For all even-numbered fold computations, CE_0 supplies the probability scores for PE computation. This is because, during even-numbered fold computations (as explained in section 4.3) all CE_0 in the PE systolic arrays are ready with their probability scores. Similarly, during all odd-numbered computation, CE_1 supplies its coefficients for computation. During computation mode, subject sequence residues flow through the SubSeqIn port to fetch the substitution matrix coefficient, $s(x_i,y_j)$, for the PE to perform the elementary operations of the alignment algorithm.

4.5.4 The OCC scheduler

The simplified state machine in Figure 4.14 illustrates the overall operations of the scheduler, which is designed in the MAIN CONTROLLER of the proposed core. This scheduler manages the fixed CEs in the PE by implementing the efficient scheduling technique outlined in section 4.3. State S0 involves configurations of all memory-based units, including the QUERY MEMORY, the SUBSTITUTION MATRIX MEMORY and the PARALLEL LOADER. Once all of these units are ready, the next state, S1, initiates for CE_0 configuration (*Initial Config*. phase). Beginning from this state, the query sequence is partitioned into several sub-sequences depending on the number of folds as discussed in section 4.5.1, and each query sub-sequence is read separately.

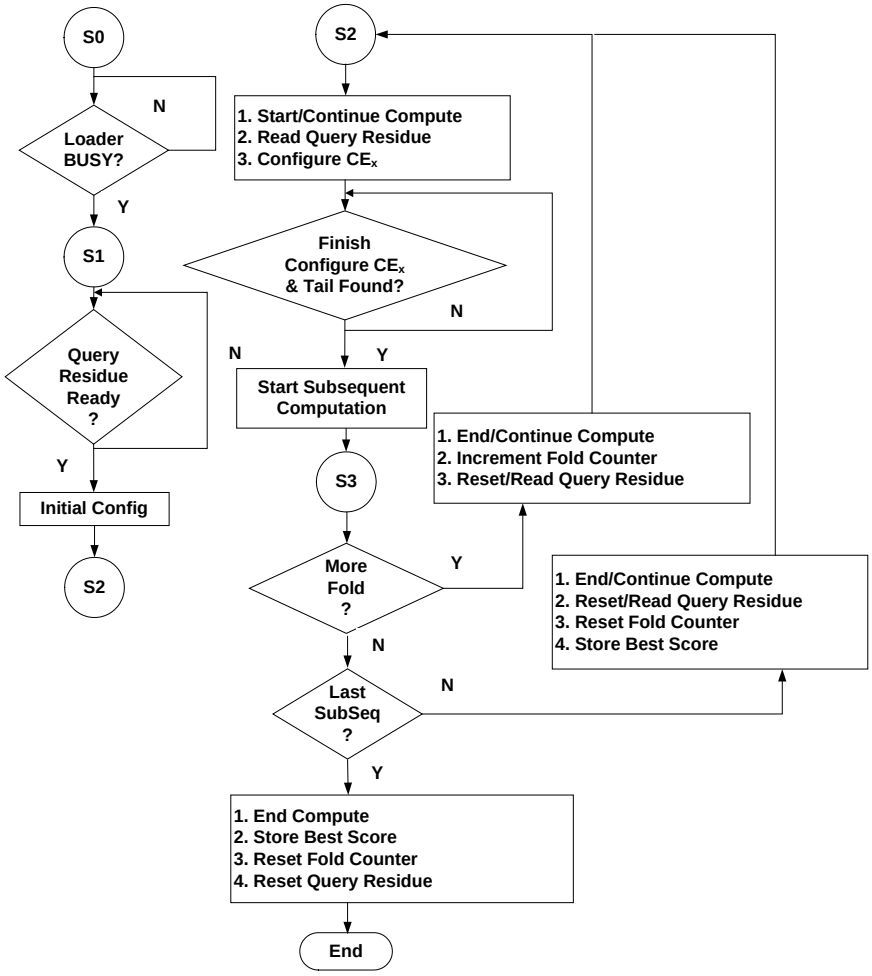


Figure 4.14: Simplified state machines of the OCC scheduler

This way, the corresponding CE can be configured efficiently so that only CE-related query residues are read from the QUERY MEMORY during configuration. Details of this mapping strategy have been discussed in section 4.5.1. The overlapping operation occurs during state S2, whereby both computation and configuration occur simultaneously. If the current configuration has finished and the pipeline is ready for subsequent fold computation (where, typically, CE configuration finishes earlier than the current computation), then the state machine moves to S3. This state decides either to continue with the subsequent overlapping operation by incrementing the fold counter or resets it if the fold counter reaches its maximum fold, to align the next subject sequence in the database. At each processing pass, the FEEDBACK FIFO stores the intermediate results before these are fed back to the input of the PE systolic arrays for subsequent fold processing. For every subject sequence that passes through the pipeline, the controller triggers the BEST SCORE FIFO to save the best score if it satisfies a given threshold value.

4.6 Implementation results

This section discusses the performance evaluation of the proposed core against the well-known SSEARCH35 ‘software only’ implementation on a standard computer and the state-of-the-art FPGA implementations. The proposed core was implemented on the Alpha Data ADM-XRC-5LX card with Virtex-5 FPGA on it. The affine gap penalty PE utilized 117 slices and, with a total of 17,280 slices, a maximum of 140 PEs can be fitted onto the XC5VLX110 device. During the hardware execution, the core was clocked at 100 MHz to search for various query sequences against a database sequence (release 2012_06 of 13-Jun-2012) which was extracted from the UniProtKB/TrEMBL database. The database comprised of 22,660,469 subject sequences or a total of 7,407,531,063 amino acids. It was stored in the host memory (consuming about 2.3GB) and transferred through the PCI bus with a data transfer rate of 2.1 Gbps. The BLOSUM 50 is used as substitution matrix to score each pair of amino acids during the alignment matrix computation and this matrix is chosen as it is the default score matrix in the SSEARCH program. Table 4.1 summarizes the overall performance of the core with varying fold factors against the SSEARCH35 program. The software was executed on the Intel Dual Core Processor, E6600 with a processor speed of 2.0 GHz. The speed-up is calculated by

dividing the software execution time by the total execution time of the proposed core. The final column in Table 4.1 clearly shows that the speed-up of the proposed core with the efficient scheduling strategy increases linearly.

Table 4.1: Total execution time and speed-up of the proposed OCC core against the SSEARCH35. The proposed core was clocked at 100 MHz and searched various length (100-2000) residues of query sequences against database of 22,660,469 subject sequences or a total of 7,407,531,063 amino acids

Query Accession	Length	# Fold	Total Execution Time(s)		Speed-up
			Proposed	SSEARCH	
P02652	100	1	91	9416	103.32
Q9H3V2	200	2	152	17160	113.00
Q8NC42	400	4	303	35992	118.70
A6NGE4	600	6	451	55704	123.50
B3KY11	800	8	599	74888	125.00
A8KA62	1000	10	766	96888	126.50
Q8NEL9	1200	12	878	112200	127.80
B2RNT9	1400	14	1067	137280	128.60
D3DNT2	1600	16	1215	157696	129.80
Q9BYP7	1800	18	1370	182864	133.50
Q12873	2000	20	1512	211024	139.60

Table 4.2 summarizes the performance of the core against other FPGA implementations. Based on the literature, the throughput for each core is reported in cell update per second (CUPS), which is a common performance indicator in computational biology. The inverse of CUPS gives the equivalent time required for a complete computation of one entry of an alignment matrix. The peak CUPS performance is determined by multiplying the number of PEs by the core's operating frequency. The proposed core comes second compared to our previously reported core. This is due to the size of the proposed PE, which is bigger than [64] in occupying two CEs in the PE, resulting in fewer PEs being fitted onto the same device. The peak CUPS performance

does not reflect the overall core performance as it does not consider data transfer and pipeline filling/flushing during alignment matrix computation.

Table 4.2: Performance comparison (in peak CUPS) against various FPGA implementations on the Smith Waterman with the affine gap penalty

Reference		**Year**	**Device**	**Slices/PE**	**PEs (#)**	**Freq (MHz)**	**Peak CUPS (Giga)**
Yamaguchi et al.	[69]	2002	XCV2000E	-	144	40	5.8
Oliver et al.	[40]	2005	XCV6000	192	168	45	7.6
Jiang et al	[65]	2007	EPS1S30	192	80	82	6.6
Benkrid et al.	[61]	2009	XC2V6000	85	168	45.6	7.66
Meng et al.	[70]	2010	XC2V6000	-	119	-	11.1
Yamaguchi et al.	[67]	2011	2V6000-4	58	168	59.3	10.0
PE with *n*CEs	[64]	2011	XC5VLX110	88	195	200.0	39.0
Proposed fixed CEs		2012	XC5VLX110	118	140	209.6	29.3

Therefore, the total execution time of each core is used to compare their performance fairly. To do so, the execution time of the proposed core is compared with that of the best reported core in [64]. Both cores were executed on the same hardware, i.e. the Alpha Data ADM-XRC-5LX card with XC5VLX110 FPGA on it, and the homology search was performed using the same query and database sequences. As a sample, both architectures were tested with different query sequences ranging from 128 residues to 2,048 residues from the protein knowledgebase. Each of the query sequences was aligned against different lengths of subject sequences in a systolic array of 128PEs with different numbers of folds. The implementation results of the respective cores are illustrated in Table 4.3. The base implementation reported in [64] requires *n*CEs to compute longer query sequences than the physically implementable PEs in the XC5VLX110 chip (where *n* is equal to the number of folds) in *n*-pass computation. On the other hand, the proposed core has only two CEs regardless of the number of folds. Thus, normalization is required in order to evaluate the speed-up performance of both cores effectively.

Table 4.3: Execution time and normalized speed-up of the proposed fixed CEs core architecture with the OCC scheduling strategy against the PE with *n*CEs core [64]. Both cores operate at 100 MHz. Input query length of 128 to 2048 residues, with each searched against a sample of 200 subject sequences

Query Accession (length)	**Q2IJ63 (128)**	**Q96B36 (256)**	**Q16515 (512)**	**Q96RT8 (1024)**	**Q8IYD8 (2048)**
#Folds	1	2	4	8	16
PE with *n*CEs [64] (*us*)	59.64	120.71	242.81	486.99	975.36
Proposed fixed CEs (*us*)	40.00	78.38	155.1	310.24	622.64
Speed-up[a]	1.49	1.54	1.57	1.57	1.57
#LCs Proposed fixed CEs	98,912	98,912	98,912	98,912	98,912
# LCs for PE with *n*CEs [64]	93780	96852	102996	115284	139860
Area Ratio[b]	1.05	1.02	0.96	0.86	0.71
Area Normalized Speed-up [c]	1.42	1.51	1.64	1.83	2.21

Therefore, a new performance metric shown in equation 4.9 is proposed here to effectively normalize performance per area of both cores. Here, LC_{ratio} is the area ratio of the two architectures. The area utilization for each core is based on the total area utilized by all PEs and memory (the FEEDBACK FIFO) in the form of logic cells (LCs) that are used to compute the alignment matrix.

$$speed\ up_{AreaNormalized} = \frac{speed\ up}{LC_{ratio}} \qquad (4.9)$$

The LC is taken into consideration as a normalization factor since it is an abstract logic resource which measures area utilization independent of the particular FPGA family's slice architecture [71]. An affine gap PE of the proposed core comprises ~117 logic slices (468 LCs, i.e. 4LCs/slice), while the FEEDBACK FIFO consumes 54Kb of BRAM. To take into account the FIFO logic resources which are used to store intermediate data between each pass computation, both the FEEDBACK FIFO and the

PEs are synthesized using Cadence Build Gates (2005) with 0.18um UMC process technology, and the gate equivalent of each is noted. From the Xilinx ISE and Cadence Build Gates synthesis results, one LC is equivalent to 443 gates and one Kbit Block RAM consumed 8174 gates respectively. Based on these two relationships, one Kbit BRAM is estimated for 18 LCs. This relationship allows for the area utilization of the PE in terms of both logic and memory resources in terms of total number of LCs. The area normalized speed-up (speed-up/logic cell) shown in Table 4.3 demonstrates that the proposed architecture has a normalized speed-up higher than 40 percent, growing linearly with number of folds.

Performing fair and meaningful comparisons against other reported FPGA implementations is difficult due to the different types of devices and families used. The use of LCs as a normalization factor, as in previous comparisons, perhaps provides a fairer evaluation. However, the different FPGA families use different lithography technologies, which may affect overall hardware performance. For instance, all Virtex-5 FPGA families were fabricated using 65 nm process technology resulting in an internal look-up table delay of 0.09ns. While all Virtex-4 FPGA families were manufactured with 90nm CMOS technology and the internal look-up table of this device is 0.17ns. Therefore, the ratio of LUT delay is taken into consideration when comparing against different FPGA families. Then, another normalization metric, the normalized speed-up/logic cell/process technology, is proposed as expressed in equation 4.10. The *LUTDelay* is the FPGA basic look-up table delay, which varies depending on the device's process technology, among other factors.

$$speed\ up_{Normalized} = \frac{speed\ up}{LC_{ratio}} \times LUT\ Delay_{ratio} \qquad (4.10)$$

Table 4.4 presents the normalized speed-up of the proposed core against other FPGA implementations. The speed-up in column A is calculated by dividing the execution time of each reference core by the execution time of the proposed ones. The normalized speed-up figures are then derived from this raw speed-up.

Table 4.4: Normalized speed-up performance (fold of 12) against the proposed core with fixed CEs

Device	Ref.	Total LCs Used	Area Ratio[1]	LUT Delay Ratio[2]	Speed-up Proposed [3] (OCC) vs. Ref.		
					A	B	C
XC2V6000	[61]	31,059	0.69	0.23	4.58	6.62	1.53
XC5VLX110	[64]	37,807	0.57	1.00	1.15	2.03	2.03
XC2V6000	[40]	43,546	0.49	0.23	5.13	10.40	2.40

The speed-up figures in column B are normalized according to their respective area consumption by dividing the raw speed-up with the ratio of logic cells (LCs) consumed by each implementation as expressed in equation 4.9. The LC ratio is determined by dividing the total logic cells utilized by the proposed core (21,458 LCs for maximum PEs of 140) by the logic cells consumed by the respective reference core. In addition, in order to normalize the speed-up according to the fabrication technology, the area normalized speed-up in column B is then multiplied by the ratio of the basic LUT delays of the FPGA technologies used following equation 4.10. The LUT delay ratio is calculated by dividing the LUT delay of the Virtex-5 XC5VLX110 FPGA with the LUT delay of each reference core. After doing this, the last column in Table 4.4 clearly shows that the proposed core is the most efficient with at least 50 percent normalized speed-up performance against others. Note that in other cases studied previously include in [72], [73], the normalized speed-up performance could not be determined due to the limited information provided, which shows the need for a standard common experimental reporting framework.

4.7 Summary and conclusions

In this chapter, a novel and efficient hardware architecture to optimize the execution time of the dynamic programming-based (DP) pairwise sequence alignment algorithm in hardware has been presented. It was realized by implementing an efficient overlapped scheduling of alignment matrix computation and the pre-loading of substitution matrix coefficients onto processing elements (PEs) in folded systolic arrays. The implementation results showed that the new hardware architecture for the Gotoh sequence alignment achieved a minimum of 103x speed-up, with the speed-up increasing linearly with the number of folds, e.g., 140x speed-up for 20-fold as compared to the equivalent software implementation. In this chapter, a new metric was also proposed to compare fairly different core implementations on different FPGA platforms. Based on the newly proposed performance metric of normalized speed-up per area and process technology (speed-up/logic cell/process technology), the designed core achieved over 50 percent normalized speed-up as compared to the state-of-the-art hardware implementation. It can be concluded that the proposed architecture with its efficient scheduling strategy has successfully optimized the execution time of DP-based pairwise sequence alignment algorithms in hardware. Among other advantages, the two great advantages of the proposed architecture over typical folded S-W architectures are summarized as follows.

Optimized Space Complexity: The proposed architecture and scheduling strategy alternately uses a fixed number of CEs (equal to 2) to compute any length of query sequences. This optimizes logic resources per PE rather than replicating look-up tables in the PE to align longer query sequences using multiple-pass computation. Moreover, the core architecture with the fixed configuration element with only two CEs in the PE enables any number of fold factors to be changed at run time.

Optimized Time Complexity: The architecture also optimizes the total execution time of the sequence homology search by virtually remove the configuration time overhead through the overlapping of alignment matrix computation and CE configuration.

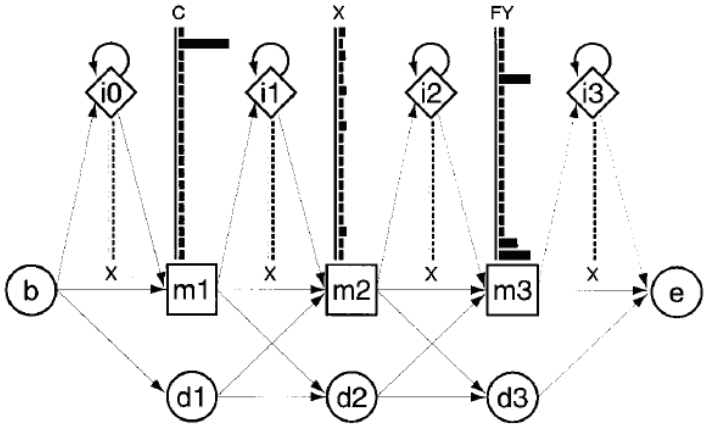
Chapter 5
Design and FPGA Implementation of the Profile HMM-based Sequence Alignment

This chapter discusses the design and implementation of the profile HMM-based sequence alignments using the well-known hidden Markov theory. Initially, the background and importance of the hidden Markov theory in sequence alignments are elaborated, before prior work on HMMER acceleration in hardware is discussed. The widely-used systolic array to accelerate the Viterbi algorithm in profile HMM sequence alignment is then presented. The discussion continues with an explanation on the use of folded systolic array architecture to align longer query profiles than physically implementable number of PEs. A novel efficient scheduling strategy and its corresponding hardware architecture are then presented. Following this, the case of recalculation and its probability of occurrence are discussed. Finally, the performance of the proposed architecture is assessed before the conclusions and possible future work are presented.

5.1 Introduction

Hidden Markov models (HMMs) have been widely-used in speech recognition applications for more than twenty years [74], [19]. In 1989, the finite state models were introduced for computational sequence analysis with the first of its applications based on DNA sequence analysis [75]. Then, HMMs were used in protein structural modeling in 1993 [76], and 1994 [77]. The general statistical modeling technique is suitable for 'linear' problems including sequences or time series [19]. The use of HMMs to model the position specific of highly similar motif in multiple sequence alignments has enabled expressions of multiple sequence alignments model explicitly in the form of profile HMMs. This has led to the extensive use of finite state models in bioinformatics including in biological sequence-to-profile alignment, whereby the specific positions of sequences in a family are modeled using the theory of hidden Markov. The probabilistic

models represent specific positions of highly conserved sequence patterns or motifs (which are sometimes also referred to as nodes) as a result of multiple sequence alignment. Motifs exist in evolutionary-related sequences. Biological sequences deviate from those of common ancestors due to the processes of mutation, selection, and genetic drift. These manifest themselves as residue substitution, deletion or insertion[3]. Profile HMMs are powerful tools for high sensitivity sequence homology searches due to their ability to scan for related sequences in a database even with low sequences identity [19]. This enables the search in a database for sequences with sparse sequence similarity, which typical pairwise sequence alignment such as the Smith-Waterman and BLAST (Basic Local Search Tool) might not be able to detect. An example of a profile HMM, which was introduced by Krogh et al. [74] in 1994, is illustrated in Figure 5.1(a). In this example, the profile HMM is constructed from the multiple sequence alignment of five sequences as shown in Figure 5.1(b). All sequences in the family share three consensus columns 1, 2, 3; each represents a highly conserved residue. The consensus columns are essentially the most prominent residues from multiple sequence alignments and hence are considered as motif positions or nodes. In profile HMM, the three columns are modeled using three match states as illustrated by the squares as labeled with m1, m2 and m3 in Figure 5.1(a).



(a)

1	2	3
C	A	F
C	G	W
C	D	Y
C	V	F
C	K	Y

(b)

Figure 5.1: (a) An example of small profile HMM representing (b).
(b) An example of a short multiple sequence alignment of five sequences [78].

Each square has 20 residue emission probabilities shown by the vertical black bars. The diamond shapes (labeled with *i0-i3*) are residue insertion states, each of which also has 20 emission probability scores. Circles labeled as *d1-d3* are 'mute' states having no emission probabilities. The main *M, I, D* states start with begin, *b* and end with *e* states, while transitions between states are shown by the arrows.

5.2 Background

In this section, the profile HMM as introduced in section 5.1 is explained in more detail. This includes descriptions of several modifications of the simple model which have been made in order to deal with real biological phenomenon. Following this, HMMER and other freely available software tools for profile sequence alignment are presented. Explanations are then given of the use of the dynamic programming-based Viterbi algorithm to search for optimal paths in profile HMM. Finally, ideas for the hardware acceleration of the computationally intensive Viterbi algorithm are presented.

5.2.1 Profile HMM with full plan 7 architecture

The profile hidden Markov model as shown in Figure 5.2 is known as the profile HMM with full plan 7 architecture. It is a modified version of the model, which was introduced by Krogh et al. as outlined in section 5.1 with several modifications made to the simple model to deal with local alignment, multiple domains and sequence fragments [19]. In this diagram, the model has four sets of match (*M*), insertion (*I*) and deletion (*D*) states, which model four specific positions of a multiple sequence alignment. The sets can be of any length, depending on the number of positions. Each *M* state represents one consensus column. All sets of *M, I, D* states are the main elements of the model and a set of *M, I, D* states is referred to as a 'node'. The insertion state is a self-transition state and multiple insertions may occur between consensus columns. State *B* (begin) and *E* (end) are the flanking states of the main model and they are non-emitting states, and hence no transition or emission scores are associated with them. The other states, *S*, *N*, *C*, *T* and *J* are special states. Both flanking and special states control algorithm-dependent features of the model, such as alignments with local or multiple-hit [79]. Local alignments are allowed by assigning non-zero state transition probabilities between the *B* state to the internal match states and from internal match states to the *E* state as illustrated by the

dotted lines in Figure 5.2. Alignment with multiple-hit occurs if the feedback score from state *J* is larger than that of state *N*. However, this case occurs very rarely and if it does, the sequence in the database usually comes from the family of the query profile HMM.

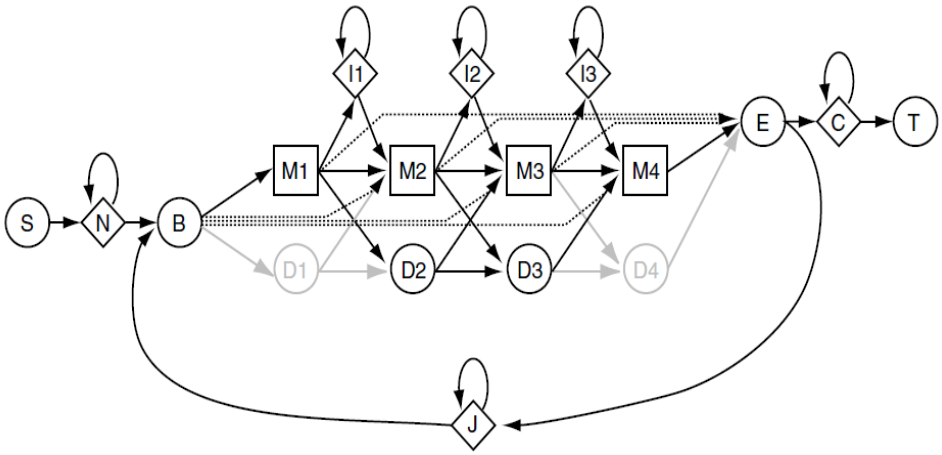


Figure 5.2: The profile HMM with plan 7 architecture [79]

A profile HMM is modeled by discrete states, whereby each represents a motif position with a probability score assigned to the state and its transitions. To understand this representation, one can imagine that an HMM generates a sequence [20]. When a state is visited, a residue is emitted from the state based on the emission probability score. On the other hand, the transition to the next state depends on any potential state with the highest transition probability score. Consequently, a transition from state to state generates the underlying state path, which is referred to as a Markov chain. In general, for a profile HMM of length L_m motif positions, the plan 7 HMM model shall comprise of L_m sets of *M*, *I*, *D* states, a set of flanking states and a set of special states. Note that, for any profile HMM length, there is neither deletion state for the first set nor deletion and insertion states for the last set. Given a sequence and a profile HMM, many state paths may potentially generate the same sequence. Only the path with the highest probability score will be chosen and this is dictated by the efficient DP-based Viterbi algorithm (the pseudo code for which is shown in Figure 5.3). The inner loop of the code comprises of three two-dimensional matrices (M, I, D), which calculate the scores of all motif positions involved in the main models for each of the residues. The outer loop

consists of flanking and special states, which are calculated at the last motif position of the query profile.

For every sequence residue *i* from 0 to *n-1*

$$N(i) = N(i-1) + tr(N, N)$$

$$B(i) = \max\begin{cases} N(i) + tr(N, B) \\ J(i-1) + tr(J, B) \end{cases}$$

$$M(i,0) = I(i,0) = D(i,0) = -\infty$$

For every model position *j* from 0 to *m-1*

$$M(0, j) = I(0, j) = D(0, j) = -\infty$$

$$M(i, j) = e(M_j, S[i]) + \max\begin{cases} M(i-1, j-1) + tr(M_{j-1}, M_j) \\ I(i-1, j-1) + tr(I_{j-1}, M_j) \\ D(i-1, j-1) + tr(D_{j-1}, M_j) \\ B(i) + tr(B, M_j) \end{cases}$$

$$I(i, j) = e(I_j, S[i]) + \max\begin{cases} M(i-1, j) + tr(M_j, I_j) \\ I(i-1, j) + tr(I_j, I_j) \end{cases}$$

$$D(i, j) = \max\begin{cases} M(i, j-1) + tr(M_{j-1}, D_j) \\ I(i, j-1) + tr(D_{j-1}, D_j) \end{cases}$$

End

$$E(i) = \max\left[M(i, j) + tr(M_j, E)\right] \quad (j = 0, \ldots, L_m - 1)$$

$$J(i) = \max\begin{cases} J(j-1) + tr(J, J) \\ E(i) + tr(E, J) \end{cases}$$

$$C(i) = \max\begin{cases} C(j-1) + tr(C, C) \\ E(i) \end{cases}$$

End

$$T(S, M) = C(N) + tr(C, T)$$

Figure 5.3: Pseudocode of the Viterbi algorithm

5.2.2 Software tools for profile HMM sequence alignment

HMMER [3], SAM (Sequence Alignment and Modeling system) [80] and PFTOOLS [81] are examples of a new generation of profile HMM software tools with 'Plan 7' model architecture [78]. In this section, discussion focuses on the HMMER package since it is a widely-used software tool for profile HMMs sequence alignment. HMMER was introduced by S.R. Eddy in 1998. The latest version, HMMER 3.0 has been ready

for use since 2010. It comprises of four programs whose functionalities are summarized in Table 5.1. In this work, the *hmmsearch* program is used for performance comparison with the proposed hardware. The program performs biological sequences-to-profile alignment. Prior to alignment, the *hmmsearch* program requires an input file in the form of a profile HMM format (.hmm).

Table 5.1: Functionality programs in HMMER 3 Package [82]

Hmmbuild	Construct a profile HMM from multiple sequence alignment
Hmmsearch	Search a profile HMM against sequences in a database
Hmmscan	Search a sequence against a profile HMM database
Hmmalign	Multiple alignment of many sequences to a common profile HMM

Then the *Hmmbuild* program can be used to construct the corresponding profile HMM from a multiple sequence alignment input file. The *Hmmbuild* program accepts either Stockholm or aligned FASTA alignment formats for this conversion. Figure 5.4 illustrates an example of a multiple sequence alignment input file with the Stockholm format. Alternatively, raw HMM files in the profile HMM format can also be accessed online from the European Bioinformatics Institute (EBI) web page. EBI has provided a collection of multiple sequence alignments and phylogenetic trees in their online database known as PANDIT (Protein and Associated Nucleotide Domains with Inferred Trees). The PANDIT database is developed based on the collection of protein families and domains available from the *Pfam* (Protein Family) database [83]. Both *Pfam* and PANDIT are developed and maintained by the EBI.

```
# STOCKHOLM 1.0

HBB_HUMAN   ........VHLTPEEKSAVTALWGKV....NVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPKVKAHGKKVL
HBA_HUMAN   .........VLSPADKTNVKAAWGKVGA..HAGEYGAEALERMFLSFPTTKTYFPHF.DLS.....HGSAQVKGHGKKVA
MYG_PHYCA   .........VLSEGEWQLVLHVWAKVEA..DVAGHGQDILIRLFKSHPETLEKFDRFKHLKTEAEMKASEDLKKHGVTVL
GLB5_PETMA  PIVDTGSVAPLSAAEKTKIRSAWAPVYS..TYETSGVDILVKFFTSTPAAQEFFPKFKGLTTADQLKKSADVRWHAERII

HBB_HUMAN   GAFSDGLAHL...D..NLKGTFATLSELHCDKL..HVDPENFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVANAL
HBA_HUMAN   DALTNAVAHV...D..DMPNALSALSDLHAHKL..RVDPVNFKLLSHCLLVTLAAHLPAEFTPAVHASLDKFLASVSTVL
MYG_PHYCA   TALGAILKK....K.GHHEAELKPLAQSHATKH..KIPIKYLEFISEAIIHVLHSRHPGDFGADAQGAMNKALELFRKDI
GLB5_PETMA  NAVNDAVASM..DDTEKMSMKLRDLSGKHAKSF..QVDPQYFKVLAAVIADTVAAG.........DAGFEKLMSMICILL

HBB_HUMAN   AHKYH......
HBA_HUMAN   TSKYR......
MYG_PHYCA   AAKYKELGYQG
GLB5_PETMA  RSAY.......
//
```

Figure 5.4: Example of multiple sequence alignment input file in Stockholm format [82]

Figure 5.5 shows the database of profile HMMs according to their length distributions. The length of a profile HMM is represented in the form of a motif which is sometimes referred to as a node. Each node represents the specific positions of multiple sequence alignments, and is used to model the profile HMM as described in section 5.1.

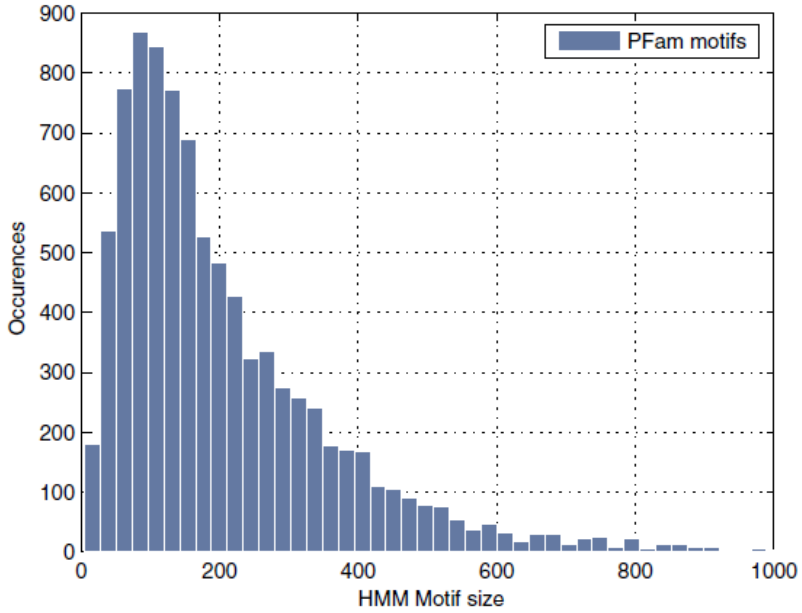


Figure 5.5: Profile HMM length (number of nodes) in the *Pfam* database [84]

HMMER 3.0 is the latest version of the profile Hidden Markov package. The most noticeable improvement in the new package compared to the HMMER 2.0 is the speed performance which is about ~100x faster than HMMER 2.0 [85]. This is due to the pre-filtering stage before the execution of a standard P7Viterbi algorithm [86]. This heuristic filter is known as the *Multi ungapped Segment Viterbi* (MSV) [86] and it leads to significant speed improvement of the *hmmsearch* program. However, the P7Viterbi algorithm is also needed for acceleration since it accounts for the overall execution time of the *hmmsearch*. This leads to the need for the acceleration of the Viterbi algorithm in hardware, where PE systolic arrays are typically used to accelerate the most time consuming part of the *hmmsearch* program. Before presenting the proposed hardware architecture to accelerate the DP-based Viterbi algorithm, prior work on HMMER acceleration in hardware is discussed in the following section.

5.3 Prior work on FPGA-based biological sequence-to-profile alignment

Biological sequences-to-profile HMM alignment is performed by aligning a profile HMM against subject sequences in the database using the well-known Viterbi algorithm. This dynamic programming-based algorithm has quadratic time complexities, when searching using a standard microprocessor. Since the successful completion of the Human Genome Projects (HGP) in 2003, an enormous number of biological sequences have been reported, resulting in an exponential increase in both numbers of protein families and the size of databases. This has led to a tremendous growth in research, which focusing on accelerating DP-based algorithms, including the Viterbi algorithm in profile HMM-based sequence alignment. HMMER is accelerated in parallel architectures, most notably using linear single instruction multiple data (SIMD) arrays and systolic arrays. Both have been proven to be good candidates for fine-grained parallel architectures for the acceleration of sequence alignment with DP-based algorithms [55], [87] and [88]. Coarse-grained parallelism is another approach, where computations of DP-based algorithms are distributed over networks of workstations. Although coarse-grained parallelism significantly increases computation performance as reported in [89], [90] and [91], such implementations consume significant amounts of power as well as involving increased maintenance and operational costs. On the other hand, parallelization using systolic arrays has also been reported for both FPGA and ASIC platforms. The latter implements systolic arrays in a special-purpose chip and has successfully provided relatively good area/performance ratios as reported in [92]; however, the special purpose hardware lacks the re-programmability which is crucial for sequence alignment. Over recent decades, FPGAs have becoming a viable alternative to the expensive and large power consumption of supercomputers and networks of workstations. The impressive speed-up of FPGAs in accelerating bio-computing algorithms has led to such reconfigurable computing platforms being consistently used as acceleration platforms for scientific computing, including for HMMER acceleration.

Early reported work of HMMER acceleration on FPGAs were presented by Maddimsetty et al. [93] and Oliver et al. [94]. The reported FPGA-based HMMER accelerations simplified the full plan 7 architecture by neglecting the feedback loop *J*, leading to an efficient fine-grained parallel architecture of systolic arrays and yielding

estimated speed-up performance of one to two orders of magnitude. However, alignment without dependency of *J* state leaves no multiple-hit detection. This result in less accurate alignment scores, especially for sequences that are closely related to the query profile. Other reported FPGA implementations with no feedback loop dependency have also been reported in [95], [96], and [97]. Although considering the *J* state guarantees more accurate alignment scores, it requires quadratic time complexity. This is because only one cell can be calculated per processing step. This impedes the anti-diagonal computation of the Viterbi algorithm in systolic arrays.

Oliver et al. then published studies of HMMER acceleration with full plan 7 architecture in 2007 [98] and 2008 [99]. A different approach was now used to calculate the alignment matrix by computing cells in the DP matrix in row-major order. This computing strategy successfully avoided the severe loss in sensitivity found in the previous work and enabled the earlier detection of the feedback path at the end of each row computation. However, the proposed strategy was not suitable for parallel computation due to feedback loop dependency. Moreover, the processing element (PE) was designed with all elements of the Viterbi algorithm, including the feedback loop, implemented in the PE. This resulted in higher slices utilization per PE of 451 logic slices. The overall core architecture was implemented on the low-cost Xilinx Spartan-3 XC3S1500 FPGA and the core achieved peak performance of 700 MCUPS with maximum numbers of PEs up to 10. In 2009, speculative computations of the DP alignment matrix to enable the acceleration of the full plan 7 HMM were reported by Sun et al. [100] and Takagi et al. [101]. This approach computes the alignment matrix in the column search space, with the feedback loop is considered when necessary. Computing the alignment matrix speculatively enables an efficient parallelism of the processing elements due to the very low tendency of alignments to occur with feedback paths. For instance, Takagi et al. [101] from the University of Tsukuba, Japan, implemented HMMER acceleration on the XC4VLX160 FPGA with the full plan 7 HMM architecture and used 100PEs to accelerate the DP algorithm. Unlike Oliver et al. [98], [102], Takagi et al. only implemented the inner loop of the Viterbi algorithm in the PE, resulting in the lower utilization of CLB slices per PE of 342 with a peak core performance of 11.8 GCUPS. In addition, an empirical analysis was carried out to verify the significance of the feedback loop in the profile HMM searches. It was reported that only 0.01 percent of the feedback loop was selected during the alignment of sequences

with a total length of 10,475,117,170 residues. The very low tendency for feedback path selection has enabled the parallelization of the algorithm without compromising computational accuracy. Takagi et al. also presented two different ways to implement parallel processing in systolic arrays, as illustrated in Figure 5.6. The search can be performed by scanning either down the search space, as in Figure 5.6(a) or along the profile HMM nodes, as in Figure 5.6(b). The authors suggested that, the first method is more suitable for speculative computation due to the very low tendency for the feedback path selection. This search strategy enables full parallelism of the PE systolic arrays and only considers the feedback loop when necessary. On the other hand, method 2 triggers the decision for recalculation faster than method 1 as it scans along the profile HMM nodes.

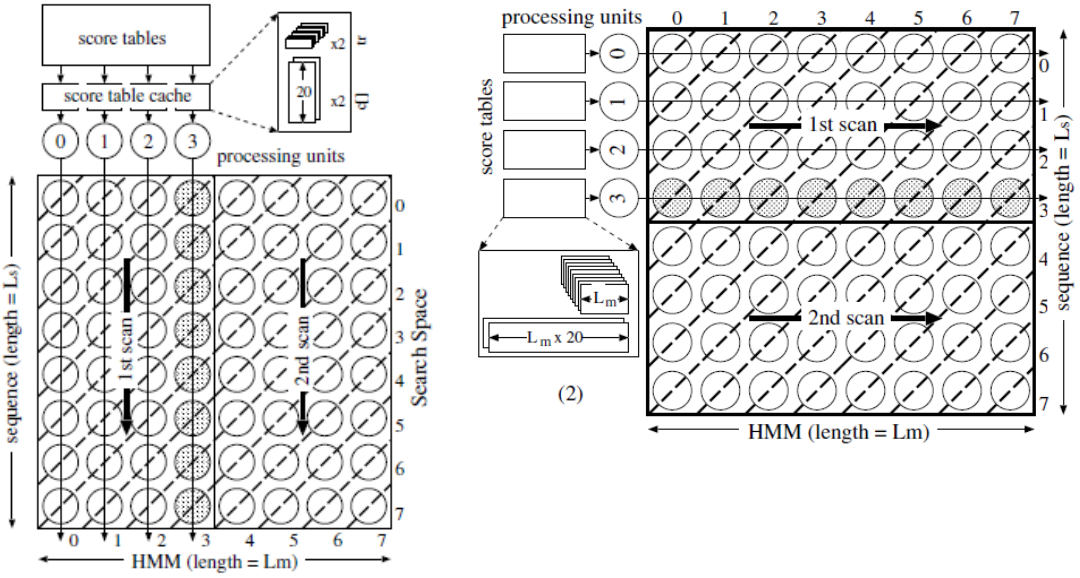


Figure 5.6: (a) Method 1: Four processing units scan down the search space along the subject sequence (b) Method 2; Four processing units scan along the profile HMM nodes.

In 2010, another approach was proposed by Derrien and Quinton [84] from the Centre de Recherché INRIA René, France. The authors used the same idea as Oliver at al. [98], however the parallelization scheme was somewhat more sophisticated. It was based on polyhedral space-time transformation which allows for the derivation of a simple and parallel architecture for HMMER acceleration. The core was implemented on

the XC3S4000 FPGA with the maximum number of PEs of 32. In the case of a motif length of 250, the core yielded a speed-up performance of 70x as compared to the HMMER 2.3.2 which ran on Intel Pentium 4 processor with 3 GHz clock frequency. Although the proposed parallel strategy addressed the aforementioned shortcomings, the scalability of processing elements is still somewhat prohibitive due to the limited numbers of embedded memory blocks, where the numbers of blocks RAM increases in step function as PEs are replicated for higher performance. In the same year, from the same research centre, Abbas et al. in [86] proposed the rewriting of both the MSV filter and the P7 Viterbi algorithm of the new version of *hmmsearch* in HMMER 3.0 package in order to make them amenable for hardware acceleration. This idea involved a potentially new level of parallelism in the algorithm by rewriting the mathematical formulation. Unfortunately no experimental results were presented and only expected levels of speed-up improvement of 10x. In 2012, Juan Fernando Eusse from the University of Brasilia, Brazil, proposed another idea to speed-up the Viterbi algorithm using a divergences technique [86]. The search technique is much like the heuristic technique in the BLAST algorithm, whereby only the region of interest (here, the diagonal region of an alignment matrix) is calculated for alignment. The design was captured using VHDL in a parameterizable manner and the overall system was implemented on the Altera Stratix II FPGA. The core architecture was prototyped on the EP2S180F1508C3 chip with a maximum frequency of 67 MHz and up to 85 PE systolic arrays were implementable in the hardware. The core was compared with un-accelerated HMMER package which was executed on an Intel Centrino Duo with 1.8GHz operating frequency and resulted in a speed-up performance of 182x. Although the speed-up significantly improved, the results were, however, less sensitive due to the partial plan 7 architecture. Moreover, this architecture also had issues of PE scalability due to the use of prohibitive blocks RAM in the PE to store the emission and transition probability scores of the profile HMM.

Typical FPGA-based HMMER acceleration computes alignment scores in systolic arrays by allocating one processing element (PE) per profile HMM node. Each PE required between 300 to 500 logic slices to implement the Viterbi algorithm resulting in 10 to 100 PE systolic arrays implemented in hardware depending on the FPGA chip used. With profile HMM of an average length of about 200 nodes [103], more logic slices and a larger amount of block RAMs (BRAM) are required since PEs are replicated

to increase parallelism. Therefore, folding technique has been used to allow for longer profile HMM implementations on arbitrarily-sized FPGA chips. This technique reuses PEs to compute alignment scores through several passes. For instance, in a linear systolic array of size n_{PE} and a profile HMM of length L_m, where $L_m > n_{PE}$, a fold factor of $k=L_m/ n_{PE}$ is required. Through folded architecture, the alignment is performed in F passes over the same number of systolic arrays of size n_{PE}. For subsequent processing passes, the PE must be updated with new emission and transition probability scores (henceforth referred to as coefficients) before alignment computation starts. In addition, a feedback FIFO (First-In-First-Out) is required to temporarily store intermediate data between passes. Previous work on FPGA-based HMMER acceleration has seen the use of blocks RAM to hold coefficients for alignment matrix computation. In terms of area utilization, the configuration chain requires a proportional amount of BRAMs as the number of PEs increases. In addition, computing the alignment matrix in multiple-pass requires the serial configuration chain to update all PEs with coefficients for every fold computation. This increases PE configuration time by a factor of k, where k is the number of folds.

Alternatively, a novel hardware architecture is proposed in this research. It has a fixed number of CEs (equal to 2) in the PE to hold coefficients for alignment matrix computation. Moreover, the CE is implemented using abundant FPGA logic slices. This reserve the restricted blocks RAM for other crucial tasks in profile HMM-based sequence alignment. In addition, an efficient scheduling strategy between alignment matrix computation and CE configuration is implemented into the core to effectively manage the fixed number of CEs. This optimizes area (logic and memory) resources as well as reducing overall time complexity. Another attractive feature of this architecture includes the fact that the computational parameters such as the number of folds and input profile HMMs which can be changed at run time. Details of the hardware design and the corresponding implementation are discussed in the following section.

5.4 The proposed hardware implementation

This section presents the design and implementation of the HMMER acceleration in hardware with a new architecture based on the double buffering technique. To achieve better scalability of the PE systolic arrays, the PE is designed to be independent of block RAM resources in FPGAs. The discussion starts with an explanation of HMMER acceleration using systolic arrays in multiple-pass computation. The double buffering technique, which is used to efficiently manage operations between computation passes, in then described in the context of profile HMM-based sequence alignment. Then, a detailed explanation of the system architecture is given including the case of recalculation.

5.4.1 Parallelizing the Viterbi algorithm and processing it in multiple-pass computation

The systolic array is a widely-used technique to exploit parallelism in FPGAs. In hardware, the recursive operation of the DP-based Viterbi algorithm is divided into smaller sub-problems, and these are then computed in parallel using systolic arrays. Given L_m as the length of a protein family and L_s as the length of the subject sequence, computing this recursive equation using a linear systolic array of size $L_m \times L_s$ results in a time complexity of $O(L_m + L_s - 1)$. Figure 5.7(a) illustrates the advantage of systolic array-based computation (shown by the anti-diagonal dotted lines in the alignment matrix). In actual hardware implementation, typically $L_m > n_{PE}$, and therefore, L_m is computed in several passes by re-using the PE systolic arrays. The intermediate results from each computation are then stored in a FIFO, as illustrated in Figure 5.7(b). In the proposed core, the PE temporarily stores emission and transition probability scores in a look-up table (referred to as a configuration element (CE)) rather than a block RAM. This look-up table is inferred from the abundant CLB logic slices. Two CEs are used in the PE to enable smooth transitions between computation passes, where as one CE is used for the current computation, the other will be updated with new emission and transition probability scores for subsequent pass computation. The mapping of the CE and its corresponding motif position is illustrated in Figure 5.7(a). In this example, L_m=16 and n_{PE} = 4. Therefore, four multiple-pass computations (k=4) are required to compute motif length of 4 n_{PE} (where only n_{PE} is implementable in the FPGA). Prior to the computation

of the alignment matrix, the CE_0 in the PE is configured with its corresponding motif position as illustrated by the CE-MOTIF MAPPER unit in Figure 5.7(a). During each computation pass, the intermediate results for subsequent fold computation are stored in a FIFO called the FEEDBACK FIFO.

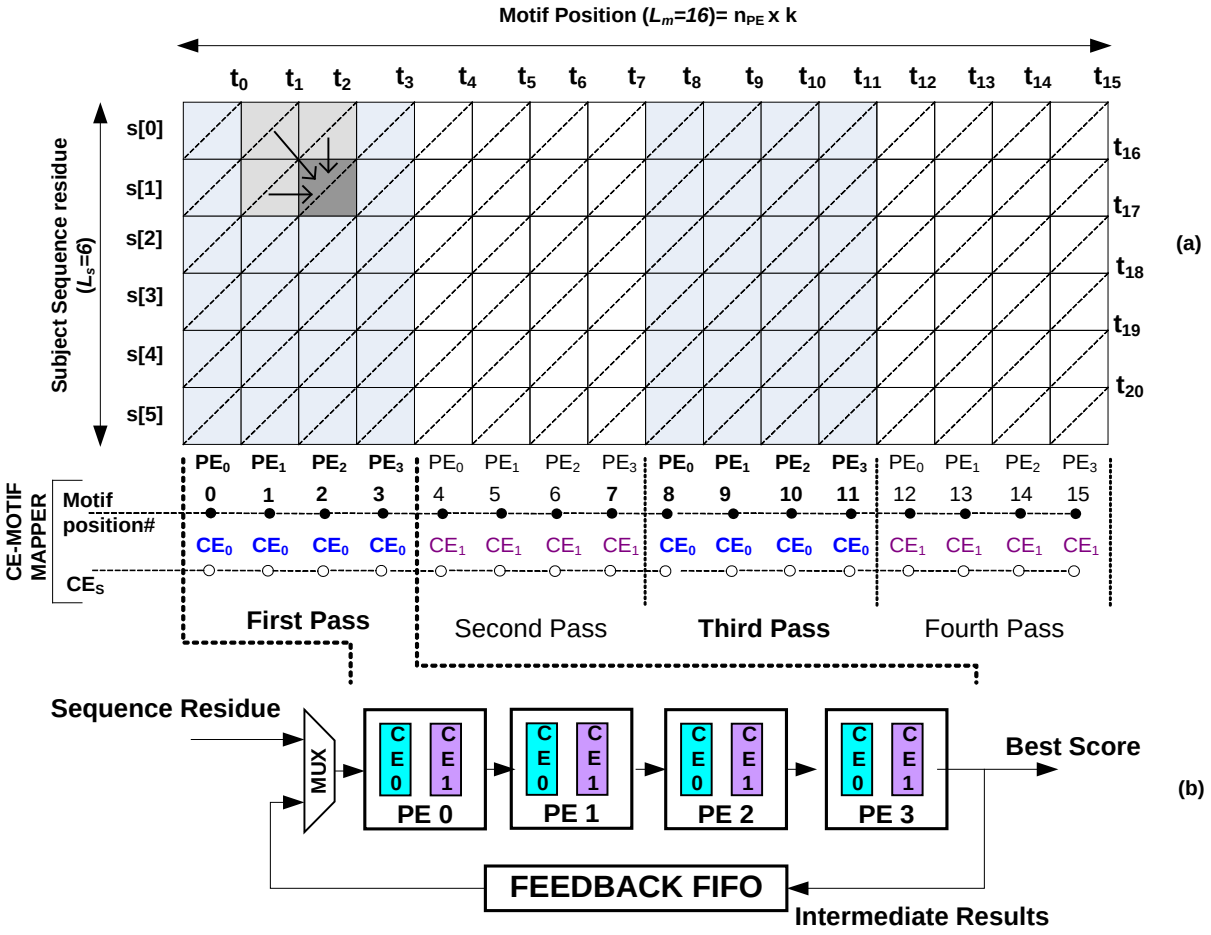


Figure 5.7: (a) DP-Alignment matrix with L_m=16 and L_s =6. (b)Parallelizing the Viterbi algorithm in four passes computation

The use of fixed CEs optimizes CLB slice utilization per PE. This is because proportional amount of logic slices are required due to the PE replications. Therefore, the PE is made smallest possible to optimize the area so that more PEs could be implemented in hardware so as to give higher computational performance. To efficiently manage the fixed CEs and ensure smooth transitions between computation passes, an efficient scheduling strategy based on the double buffering technique is used. This way,

both tasks; alignment matrix computation and CE configuration occur concurrently, thus significantly reducing the time complexity needed to configure the CE. A detailed explanation of this scheduling strategy is given in the following section.

5.4.2 The efficient scheduling strategy for alignment matrix computation and CE configuration

The efficient scheduling strategy implemented in the core architecture is illustrated in Figure 5.8. It is adopted in the core architecture in order to efficiently manage the fixed CEs for the concurrent operation of alignment matrix computation and CE configuration without interrupting the ongoing alignment matrix computation. In this work, this is referred to as overlapped computation and configuration (OCC) whereby the task of computing the alignment matrix overlaps with CE configuration, which is labeled as *Overlap* in Figure 5.8. This way, the configuration time is virtually removed, thus optimizing the total execution time of the DP-based Viterbi algorithm. This illustration is based on the explanation in section 4.4, where computation proceeds in four passes due to the fact that only n_{PE} could be implemented on the FPGA device in hand. To allow for efficient scheduling, all CE_0 elements in the PE pipeline are initially configured with coefficients during the *Initial Config*. phase. This is the only non-overlapping configuration operation. Once the first pass (F_1) computation starts, the CE_1 in the pipeline is updated with new coefficients for subsequent fold computation (labeled as *Overlap 1*).

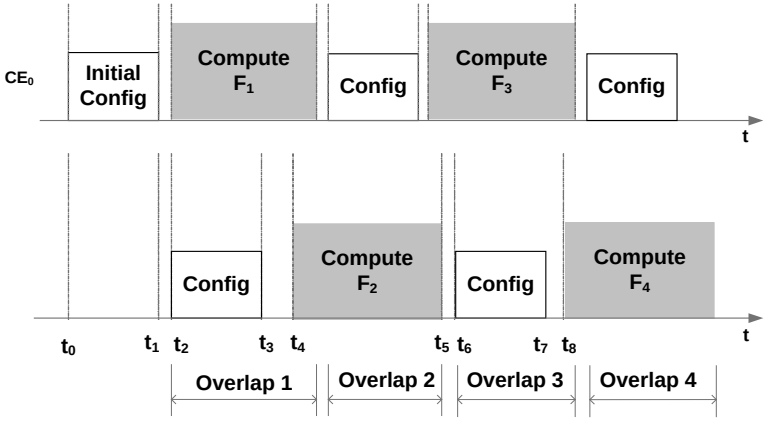


Figure 5.8: Efficient scheduling strategy between alignment matrix computation and CE configuration.

This overlapping operation continues until all subject sequences in the database are exhausted. Note that, during the *Overlap 4*, CE_0 is updated with new coefficients for the next subject sequence. Each sequence is delimited by a special character to mark the start and end of a new subject sequence.

5.5 The novel system architecture

In this section, the system architecture is described in more general terms before details of each unit are elaborated in subsequent sections. The core architecture of the Viterbi algorithm as outlined in section 5.2.1 is illustrated in Figure 5.9. The PE_BLOCK essentially comprises of a number of parameterizable basic processing elements (PE_i) where each PE has fixed configuration elements. PE systolic arrays accelerate the alignment matrix computation of the DP-based Viterbi algorithm using the overlapping computation and configuration strategy. The mapping of a CE to its corresponding motif position as discussed in section 5.4.1 is dictated by a controller inside the CE-MOTIF MAPPER. The local controller keeps both CEs busy which either hold probability scores for alignment matrix computation or update the CE with new scores for subsequent computation. The controller sets all CE_0 elements in the pipeline to hold probability scores during even-numbered pass computations, whereas CE_1 elements are used for all odd-numbered pass computations. Any CE which is not currently used for computation is updated with coefficients for subsequent pass computation. During multiple-pass processing, the FEEDBACK FIFO temporarily stores intermediate results from each pass before they are fed back to PE_0 through the input multiplexer. The depth of the FIFO is dictated by the length of the subject sequence. The BEST SCORE FIFO stores the offset addresses of subject sequences and their corresponding scores which satisfy a predefined threshold value. Note that a special unit, the *Recalc.Unit*, is designed inside the core. This unit monitors the feedback score of every sequence residue at the last motif position and triggers the core to go into recalculation mode whenever the score from the feedback loop is dominant. A Recalculation FIFO temporarily stores all *M(i,j)*, *I(i,j)* and *D(i,j)* scores for each PE, where the input from the PEs is selected by an n_{PE} to 1 multiplexer. The MAIN CONTROLLER is a scheduler for the OCC operation. It manages the two CEs by alternately using them for computation and configuration depending on the aforementioned computation passes. For instance, while CE_0 holds the

coefficients for alignment matrix computation, CE_1 will be configured with new coefficients for subsequent fold computations, and vice versa. This way, both computation and configuration modes run simultaneously, thus optimizing total execution time by reducing the configuration time overheads as a result of the overlapping operation. In addition, the same systolic array (PE_BLOCK) is reused in the case of multiple-pass alignment matrix computation without requiring additional logic resources.

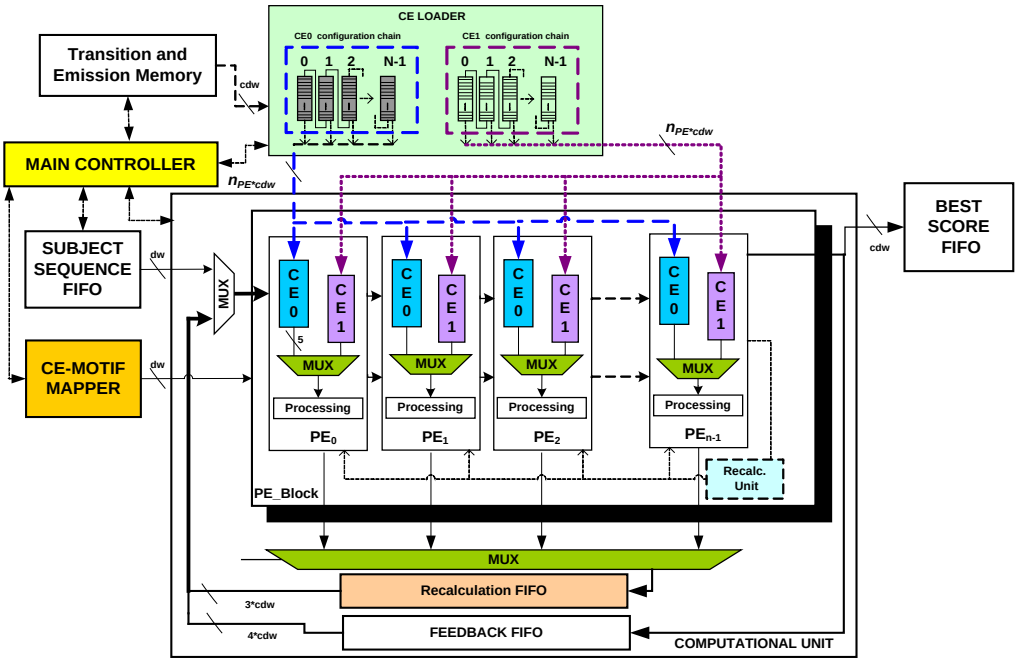


Figure 5.9: The system architecture for the full plan 7 HMMER accelerator

5.5.1 The processing element (PE)

Figure 5.10 depicts the simplified internal architecture of the proposed PE. It has two configuration elements, CE_0 and CE_1, which temporarily store emission and transition probability scores of a particular profile HMM motif or node. The PE is designed with zero dependency on the restricted block RAM resources. Its main task is to find the optimal path of the Viterbi algorithm by means of calculating scores for the DP-based algorithm. It comprises of two CEs which are used alternately for alignment matrix computation, where the turn for the CE (either CE_0 or CE_1) is dictated by computational

passes as outlined in section 5.5. During alignment matrix computation, the CE supplies both emission and transition probability scores as input for the 'processing' engine. The configuration element consists of three look-up tables; 1) 20 elements emission scores of the *M* state 2) 20 elements emission scores of the *I* state and 3) 9 elements of the transition state scores. Each CE, with a CE_{Depth} of 49 elements (the total depth of all three look-up tables) represents a particular motif position.

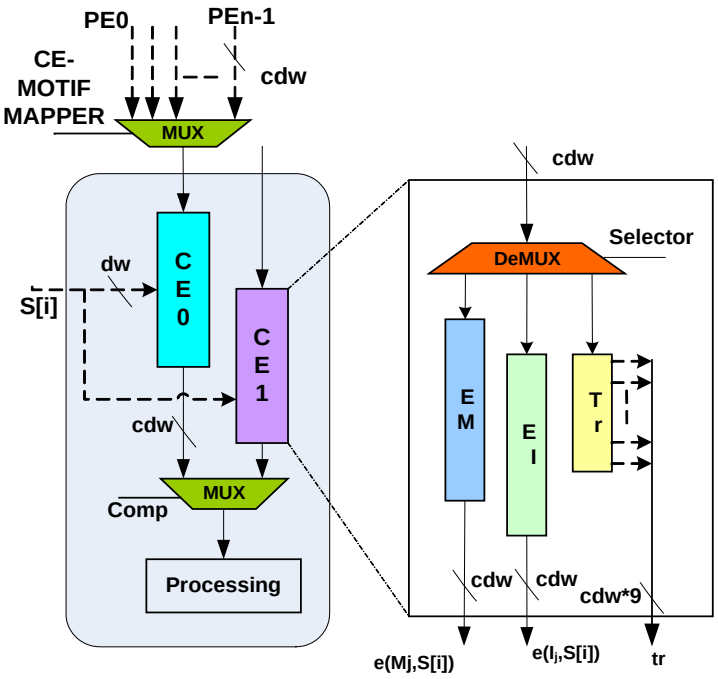


Figure 5.10: The proposed PE with two configuration elements (CEs) with each CE holds emission and transition probability scores

The 'processing' part, which implements the inner loop of the Viterbi algorithm and its corresponding logic operation which realizes the algorithm in hardware, is shown in Figure 5.11. It implements the elementary operations of the Viterbi algorithm which give the two-dimensional matrices *M(i,j)*, *I(i,j)* and *D(i,j)* as described by pseudo code in section 5.2.1. The scores for these matrices are calculated in parallel and their output is delayed by one clock cycle, so that *M(i-1,j-1)*, *I(i-1,j-1)* and *D(i-1,j-1)*. To illustrate the data dependency among PEs, consider PE_2 in Figure 5.7 as an example, where the systolic operation computes the alignment score of residue *s[1]* at t_3. These data dependencies require outputs of the previous PE (i.e. PE_1) from t_1 and t_2 for the upper-

left and left dependencies of the PE respectively. The PE also requires its own output from the previous processing step, t_2. The left, upper and upper-left dependencies of the *M*, *I* and *D* cells are then fed into PE_2 for the computation of its alignment score.

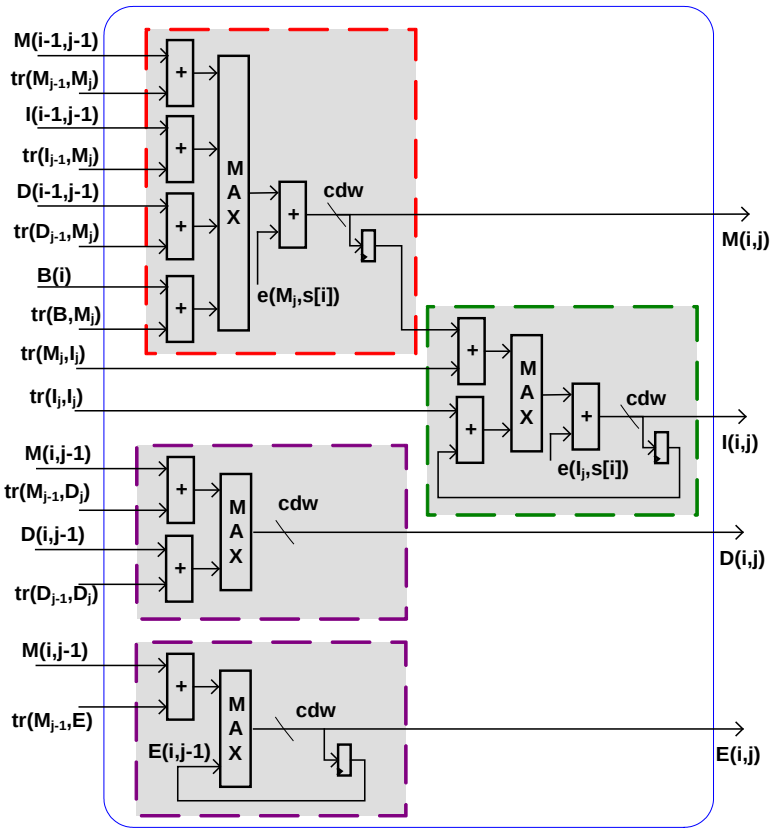


Figure 5.11: The processing engine inside the PE

The *E(i,j)* instance computes scores of the *E* state, either from the most probable path that arrives at state *E* with transition from state *M* or the score of the path that ends at state *E* with self-transition. It compares the maximum score from the PE_{i-1} with the current *E(i,j)* score before emitting the maximum of the two to PE_{i+1}. During each processing step the input residue *s[i]* is propagated to the subsequent PE along with the *M*, *I*, *D* and *E* scores. Ultimately, the score from E emitted by the last PE in the final processing step is the score of the alignment. Other one-dimensional matrices (*N*, *B*, *J*, *C* and *T*) are not implemented here to reduce the size of the PE. Note that, both *dw* (data width) and *cdw* (compute data width) are parameterizable and in this core, they are set to 5-bit and 15-bit respectively.

5.5.2 The CE loader

The CE Loader supplies the coefficients of a profile HMM to the PE in the form of emission and transition scores for each motif position. Two independent configuration chains, the CE_0 configuration chain and the CE_1 configuration chain are designed in the loader which is discussed in section 5.5 (see the overall core architecture in Figure 5.9). Each of these is directly connected to CE_0 and CE_1 in the pipeline PEs. This way, each CE is updated independently whenever the pipeline has finished a computation, without incurring additional delay. In this explanation, only one configuration chain is presented for the sake of simplicity in describing the internal architecture. The hardware architecture for the CE_0 configuration chain is shown in Figure 5.12.

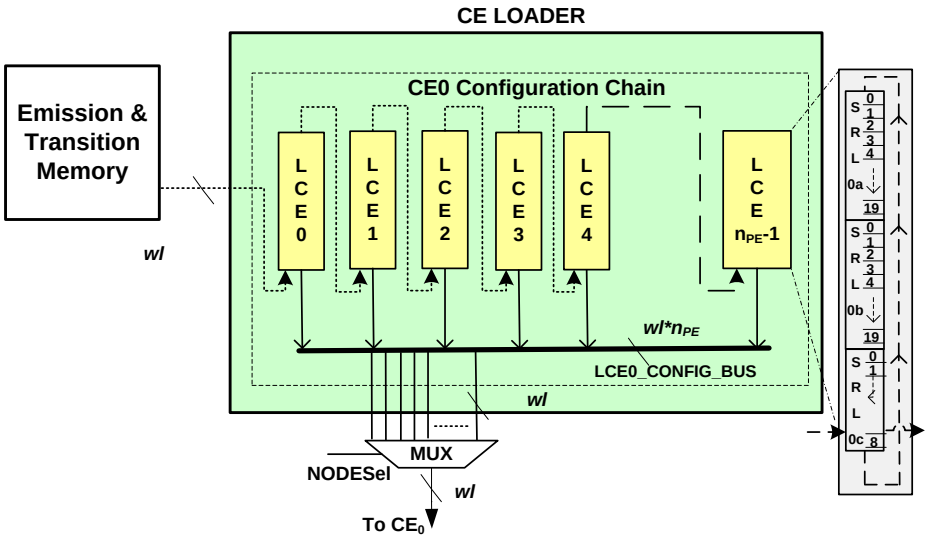


Figure 5.12: The CE loader for CE_0 in the PE

This simplified diagram also applies for the CE_1 configuration chain. The configuration chain is made up of a serial chain of circular buffers (LCE_x), which is implemented using the shift register look-up table (SRL) available in the FPGA's slices. Each buffer holds a profile HMM node, which comprises of 49 elements (i.e. CE_{Depth} = 49) of the emission and transition probability scores. The loader has two modes; the initial configuration mode and the operational mode. During initial configuration mode, all profile elements in a profile HMM are shifted serially. The mapping between a buffer and its corresponding profile HMM node is the same as for a CE to a profile HMM node as mentioned in section 5.4.1. Once all coefficients are successfully loaded, the loader

switches to the operational mode. During this mode, these buffers constantly cycle all coefficients of profile HMM emission and transition probability scores, presenting to the PEs complete columns of the corresponding motif positions at every multiple of CE_{Depth} clock cycles. Synchronization pulses are emitted at every CE_{Depth} clock cycles to mark a valid duration for coefficients loaded into the CE. This way, all CEs are configured simultaneously, with a worst case configuration time of $2CE_{Depth}$ clock cycles, i.e. $tCE_x \leq 2CE_{Depth}$. The total configuration time, taking into account the loader's initial configuration time, is defined by equation 5.1.

$$T_{config} = tLCE_x + tCE_x \qquad (5.1)$$

where $tLCE_x = n_{PE} \times CE_{Depth}$. This is the initiation configuration time needed to update the configuration chain with new coefficients. In the case of aligning longer profile HMMs in multiple-pass computation, the remaining configuration time is virtually removed as a result of the overlapping computation and configuration strategy.

5.5.3 The case of recalculation: roll back computation

This section discusses the case of recalculation for the full plan 7 HMM architecture. In most cases, the Viterbi algorithm could be calculated speculatively in parallel by eliminating the dependency of the *J* state for the subsequent residue.

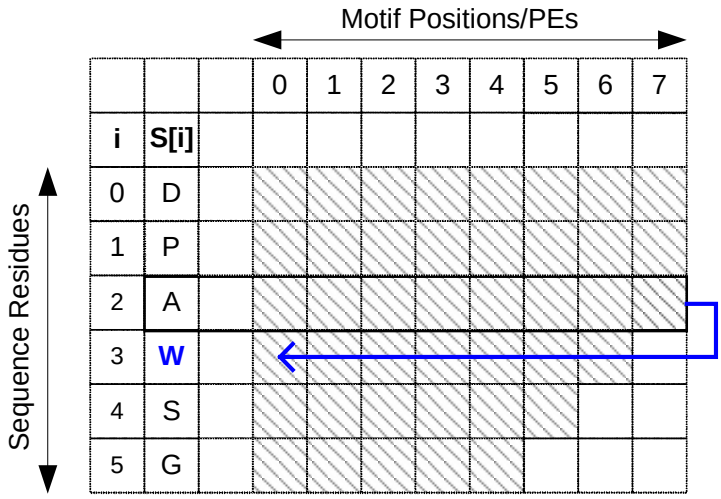


Figure 5.13: The case of recalculation beginning from residue '*W*'

This enables full parallelism in alignment matrix computation. Recalculation is considered when necessary and, in the worst case, it may occur L_s*-1* times. In this case, computation turns to sequential fashion. In the diagram shown in Figure 5.13, a profile HMM has a length L_m=8 positions and a subject sequence length L_s =6 residues. Initially, the alignment matrix is speculatively calculated. As residue '*A*' enters the last motif position at PE_7 of the profile HMM (assuming the number of PEs equals to the profile's length), the *J* score is larger than *N(i-1)+tr(N,N),* causing the alignment of the subsequent residue, '*W*', with a feedback score. Consequently, all PEs beginning from PE_0 recalculate their new *M(i,j), I(i,j), D(i,j)* scores. The recalculation starts from row '*W*' with boundary values taken from all previously computed *M(i,j), I(i,j), D(i,j)* scores of residue '*A*', which is stored in the Recalculation FIFO.

5.5.4 The main controller

This section describes the operations of the main controller for both the speculative and recalculation modes of the Viterbi algorithm. The overall controller's operation is illustrated in Figure 5.14.

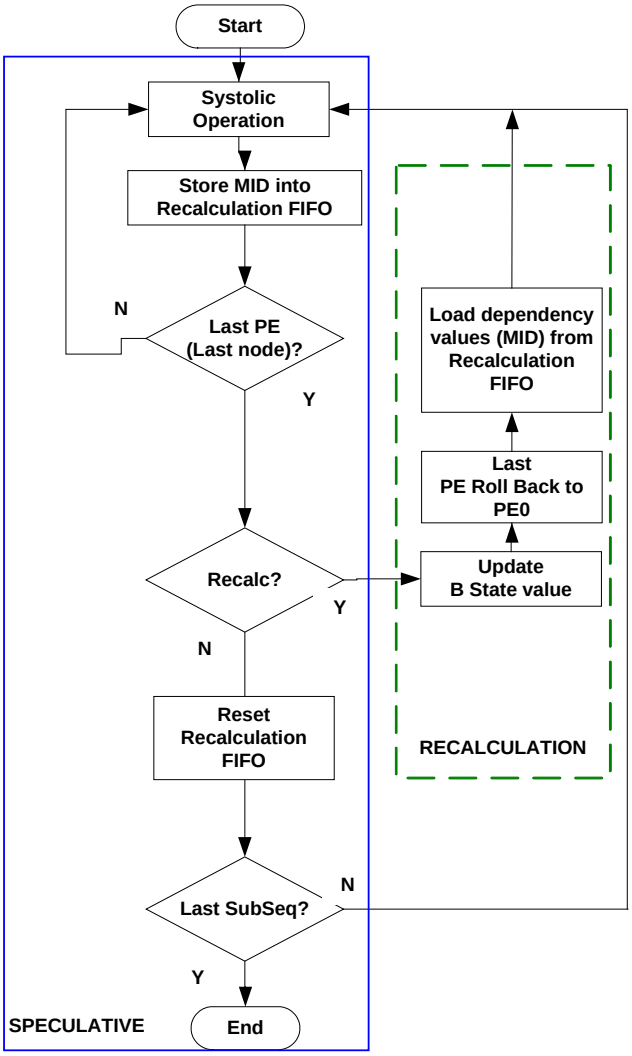


Figure 5.14: The simplified diagram of main controller for both speculative and recalculation modes of the HMMER acceleration.

During the normal operation, i.e. the speculative computation, the subject sequence from the database which is stored in the host computer is transferred to the core. The Subject Sequence FIFO in the core, buffers the incoming sequence during alignment matrix computation and the offset addresses of the selected subject sequences together with

their corresponding scores are stored in the Best Score FIFO. Recalculation can only be detected during the computation of the last node of the profile HMM. Thus, the Recalculation FIFO pre-stores the PE *M*, *I*, *D* scores as alignment matrix computation commences. The dependency values of these PE systolic arrays are required during the recalculation mode. The controller resets this data from the FIFO during the computation of the last node if no recalculation flag is triggered by the Recalculation Unit. This cycle continues with the controller scanning for the recalculation flag at the end of the profile HMM node, and if there is a need for recalculation, the controller stop reading new residues from the subject sequence FIFO and PE computation rolls back to PE_0 to start the recalculation mode. The dependency values of the pre-stored *M*, *I*, *D* scores are loaded into their corresponding PE systolic arrays before recalculation commences. All PEs will be updated with the pre-stored *M*, *I*, *D* scores. Similarly, the score for state *B* is updated with the score from state *J* instead of *N* following the recalculation procedure, as outlined in section 5.5.3. This cycle continues for one residue after another until all subject sequences in the database are exhausted.

5.6 Implementation results

In this section, the implementation results for the HMMER acceleration on the Alpha Data ADM-XRC-5LX board with the Virtex-5 FPGA are presented. The core was designed using Verilog HDL to implement the Viterbi algorithm with full plan 7 architecture in hardware. From the place and route report, the PE consumed 337 logic slices and therefore, with a total of 17,280 slices of the Virtex-5 XC5VLX110-3ff1153 FPGA, 43 PEs can be fitted onto the chip. Moreover, during the test, the computation word length of this core was set to 16-bit. Table 5.2 summarizes the hardware resources which are required to implement the overall core of the full plan 7 architecture.

Table 5.2: Hardware resource utilization of the *hmmsearch* with full plan 7 architecture (43 PEs) generated from the Xilinx ISE 13.2 place and route report

Device	Xilinx XC5VLX110-3ff1153		
Hardware Resources	**Available**	**Used**	**Utilization Ratio (%)**
Slices	17,280	14,651	84
LUTs	69,120	46,403	67
FFs	69,120	33,146	47
Block RAM (36k)	128	42	32

During the normal computation mode, the core calculates its alignment score speculatively and switches to recalculation mode whenever required. The very low tendency for recalculation to occur has enabled the full parallelization of the time-consuming DP-based Viterbi algorithm in the PE systolic arrays. To evaluate this case empirically, several samples of randomly picked profile HMMs from the *Pfam* database [103] were searched against the database of 230,150 sequences or 84,479,584 residues. The database sequences were extracted from the UniprotKB/Swiss-Prot knowledgebase[104]. From this analysis, the total number of residues involved in recalculation was 3,965 out of 84,479,584 residues or only 0.01 percent. This shows that almost all sequences in the database can be computed speculatively to enable the anti-diagonal computation of the systolic arrays. In an effort to keep the architecture of the proposed design closer to the full Plan 7 HMM architecture, as in the HMMER package,

the core has been designed to support both speculative and recalculation computation modes. The corresponding speed-up performance of the designed core compared to the latest version of HMMER i.e. the HMMER 3.0, is shown in Table 5.3.

Table 5.3: Speed-up performance of the proposed core against HMMER 3.0, PEs = 38, core operating clock frequency 150 MHz. The profile HMM length of 30 to 532 nodes was searched against a database sequence of 517,100 sequences or 182,146,551 residues

Pfam Accession #	Length (Nodes)	Fold	FPGA Execution time (s)	HMMER 3 Execution time (s)	Speed up
PF00132	38	1	0.20	6.90	34.50
PF00200	76	2	0.41	8.34	20.34
PF00250	152	4	3.26	28.27	8.67
PF00294	304	8	13.03	35.74	2.74
PF00282	380	10	20.35	40.82	2.01
PF00232	456	12	29.31	44.18	1.52
PF00183	532	14	31.18	48.99	1.57

HMMER is a portable operating system interface (POSIX)-compatible platform and thus it can only be run on either UNIX or Linux environments. In this research, the *hmmsearch* software was executed via Secure Shell (SSH), i.e. the Linux-based command interface that is connected to remote computers. The machine comprises of 16 Intel CPUs with operating frequency of 2.5 GHz each, and it has a total memory of 63.02 GB RAM. The database sequence was extracted from the UniprotKB/Swiss-Prot knowledgebase version 18 May 2012 with 517,100 sequences or 182,146,551 residues. To access samples of protein families with the hidden Markov models format, the Protein and Associated Nucleotide Domains with Inferred Trees (PANDIT) information web page was used [105]. This was developed by the European Bioinformatics Institute (EBI) for the easy access of collections of multiple sequence alignments and phylogenetic trees. In this analysis, protein families with 10 to 440 sequences were used. In the web page, multiple sequence alignments are represented by profile HMM raw

files. The collection of the profile HMM nodes ranges from 38 up to 532 nodes. During the homology search, the proposed core was clocked at 150 MHz with 38 PEs. In terms of comparison against HMMER 3.0, the proposed core with various fold factors achieves an average speed-up of 10x.

In the case of comparison against other FPGA implementations, the work previously reported in [84], [98], [102], [101], [106] and [107] were used. Table 5.4 shows the core's performance against the full plan 7 HMMER acceleration on other reported FPGAs over the last five years. Performance was measured in CUPS (cell update per second) as this is a common performance indicator used in computational biology. The inverse of CUPS gives the equivalent time required for a complete computation of one entry of an alignment matrix. The peak CUPS is determined by multiplying the number of PEs by the core's operating frequency. From Table 5.4, it is clearly shows that the proposed core is the fastest in terms of operating frequency as compared to others. More specifically, it is 41 percent higher in clock frequency than [101] and in terms of CUPS performance, it comes second fastest after [101]. This is due to the total number of logic slices available in [101] is twice as many slices (after taking into account the fact that one Virtex-5 slice equals two Virtex-4 slices) as the XC5VLX110 [108]. Although the peak CUPS figure is widely-used to evaluate the performance of systolic arrays, this depends on the number of PEs which varies depending on the type of FPGA used. Any silicon chip with more slices can implement more PEs, thus resulting in higher CUPS performance and vice versa.

Table 5.4: Performance comparison (in peak CUPS) against various FPGA implementations of the full plan 7 *hmmsearch* acceleration

Reference		**Year**	**Device**	**Slices/PE**	**PEs (#)**	**Freq (MHz)**	**Peak CUPS (Giga)**
Steven et al.	[84]	2010	XC3S4000	583	32	60.0	1.92
Yanteng et al.	[106]	2009	XC5VLX110	622	25	130.0	3.20
Takagi et al.	[101]	2009	XC4VLX160	342	100	117.9	11.80
Oliver et al.	[102]	2008	XC2V6000	451	30	70.0	2.10
Oliver et al.	[98]	2007	XC3S1500	451	10	70.0	0.70
John et al.	[107]	2007	XC3S1500	451	10	70.0	0.70
Proposed		2012	XC5VLX110	337	43	166.0	7.14

Therefore, the alternative performance indicator of speed-up is used in this analysis. Speed-up measures all overhead time, including pipelining filling/flushing as well as other FPGA communication overheads. In an attempt to fairly evaluate our core performance against other reported FPGA implementations, the speed-up is normalized with respect to area (logic and memory) and process technology. This removes the inherent advantages of the Virtex-5 FPGA used in this work, thus makes it comparable to others. The normalized performance indicator is shown in equation 5.2, which is the same as in equation 4.10.

$$speed\ up_{\ Normalized} = \frac{speed\ up}{LC_{ratio}} \times LUT\ Delay_{ratio} \qquad (\ 5.2\ )$$

where LC_{ratio} is the ratio of the total logic cells (logic and memory resources) used in the core architectures under comparison and the $LUT\ Delay_{ratio}$ is the ratio of the FPGA's Look-up Table (LUT) delays. Due to the different internal slice architecture in different FPGAs, the logic cell or LC is used in this analysis rather than the CLB logic slice. The former is an abstract logic resource which measures area utilization independent of any particular FPGA family's slice architecture. For fair and meaningful comparison, the same profile HMMs (*Pkinase*, L_m=294) was aligned against a target sequence (*Artemia*, L_s = 1405) as reported previously in [101]. The proposed core required 7 folds and its total execution time was 48.59 us when aligning *Pkinase* with *Artemia*. On the other hand, Takagi et al. [101], with more slices on the chip only required 3 folds to compute the same model. The reported execution time was 56 us. By dividing the execution time of [101] with that of the proposed core, the designed core achieved a raw speed-up of 1.15.

To evaluate the core performance fairly with that of Takagi et al., the raw speed-up is then normalized per area and process technology. This way, the normalized figure evaluates the core performance independent of different FPGA devices and fabrication technologies. However, the internal CLB architecture in FPGAs, even for the Xilinx FPGA, varies from one family to another and most notably between Virtex-5 FPGAs and the older Xilinx FPGA families. For instance, one CLB slice of Virtex-5 has twice as many slices as its predecessors [109]. This is due to the slightly more complex internal CLB architecture of Virtex-5 families as compared to the previous Xilinx FPGA families. In terms of internal slice architecture, one Virtex-5 slice comprises of four LCs,

while the previous Xilinx FPGA families had two LCs per slice. Due to the different internal slice architecture, which also varies depending on FPGA families; slice not effectively normalizing speed-up performance per unit area. This has led to the use of a smaller element inside the CLB slice, i.e. the logic cell, as a normalization factor. The use of the FPGA's abstract logic resource is reasonable since an LC in all Xilinx FPGA families comprises the same elements of a look up table (LUT), a multiplexer and a register. The LUT can also be used as distributed RAM or as a shift register [110]. In the proposed core, no BRAM element is used in the PE. Therefore the equivalent logic cells/PE is calculated by multiplying the number of slices to infer a PE by 4LCs (one Virtex-5 slice has four LCs). This gives the total number of LCs per PE of 1,348 LCs or, in the case of 43 PEs, a total of 57,964 LCs.

On the other hand, Takagi et al. implemented the profile HMM-to-sequence alignment core on the Virtex-4 XC4VLX160 FPGA. The authors utilized BRAM in the PE to hold both transition and emission scores during alignment matrix computation. Therefore, an equivalent number of LCs of the BRAM resources is taken into account prior to the normalization step. This is done by synthesizing a FIFO and a PE using both the Cadence Build Gates (2005) with 0.18um UMC process technology and Xilinx ISE 13.1, targeting two different Virtex architectures (XC4VLX160 and XC5VLX110) and the equivalent gate count of each is noted. From the synthesis results of both Xilinx ISE and Cadence Build Gates, one LC is equivalent to 443 gates and one Kbit Block RAM consumed 8174 gates. Based on these two relationships, one Kbit BRAM is estimated to represent 18 LCs. These relationships allow for the determination of area utilization of the PE in terms of both logic and memory resources in the form of total of LCs used. Table 5.5 summarizes the normalized speed-up performance of the proposed core against Takagi et al. [101].

Table 5.5: Normalized speed-up performance of the proposed core against Takagi et al. The profile HMM search for (*Pkinase*, L_m=294) against a target sequence (*Artemia*, L_s = 1405)

Device	Ref.	#PEs	#Fold	Total LCs Used	Area Ratio	LUT Delay Ratio	Speed-up Proposed (OCC) vs. Takagi et al.		
							A	B	C
XC4VLX160	[101]	100	3	127,368	1.00	1.00	1.00	1.00	1.00
XC5VLX110	OCC	43	7	57,964	0.46	0.53	1.15	2.53	1.34

In the case of [101], Takagi et al. utilized 100 PEs with a total of about 182, 18 Kbit BRAM or equivalent to 58,968 LCs to store both transition and emission scores. The total area utilization (logic and memory) in the case of 100PEs was 127,368 LCs. In terms of area ratio, the proposed core with only 43 PEs utilized 46 percent of the area than that of Takagi et al. to compute the *hmmsearch* of profile HMMs (*Pkinase*, L_m=294) against a target sequence (*Artemia*, L_s = 1405). Then, based on the calculated area ratio as in Table 5.5, the speed-up performance per logic cell is calculated by dividing the raw speed-up by the area ratio. The raw speed-up is calculated by dividing the execution time of the two studies, and the corresponding normalized speed-up per area is 2.53. However, the Virtex-5 FPGA was fabricated with 65nm lithography technology, while the Virtex-4 used 90nm. To take account of the advantages of the Virtex-5 FPGA in terms of fabrication technology, the area normalized speed-up is then multiplied by the LUT propagation delay ratio. It is noted that the Virtex-5 (LUT delay 0.09ns) has the advantage of 47 percent less LUT delay as compared to the Virtex-4 (LUT delay 0.17ns). Then the normalized speed-up per LC/process technology is calculated by multiplying the area normalized speed-up by the calculated LUT delay ratio. Based on the normalized figure, the designed core outperforms Takagi et al. with a normalized speed-up performance of 1.34. In this analysis, *Pkinase* was chosen due to the fact that its length is close to the average length of the profile hmm database. Due to limited information provided in the literature, however, the normalized performance indicator cannot be used to fairly evaluate other reported implementations.

5.7 Summary and conclusions

In this chapter, a novel FPGA-based architecture for HMMER acceleration has been presented. This profile HMM-to-sequence homology search uses the well-known Viterbi algorithm to calculate alignment scores. Typical hardware implementations of this dynamic programming-based algorithm require a proportional amount of blocks RAM to hold both the emission and transition probability scores of the profile HMM during alignment matrix computation. Thus, the acceleration of such an algorithm in PE systolic arrays is greatly affected by the restricted memory resources. In contrast, the proposed PE architecture has been designed using the abundant logic slices to hold the probability scores. In the case of multiple-pass computation, a fixed number of two CEs have been

designed into the PE, whereby CE_0 and CE_1 are used alternately to hold probability scores for alignment matrix computation. This way, the logic resources used for the PE are optimized and this allows for the scalability of the PE systolic arrays to give a higher degree of parallelism. Moreover, an efficient scheduling strategy has been adopted into the system's architecture to efficiently manage the fixed CEs during the multiple-pass computation. This is implemented by overlapping the task of computing the alignment matrix with that of updating another CE with probability scores for subsequent pass computation. This vastly reduces the CE configuration time as a result of the overlapping of tasks and consequently increases the overall performance of the core. An additional attractive feature of, this architecture is that it also enables the user to change the number of folds factor at run time to suit different lengths of the profile HMM.

The core architecture was implemented on the XC5VLX110 FPGA and several tests were conducted to evaluate the core performance against the corresponding software implementation as well as comparing it with other FPGA implementations. The implementation results showed that the proposed core achieved an average speed-up of 10x compared to the latest HMMER package (version 3.0). In terms of CUPS performance the proposed architecture gained peak performance of 7.14 GCUPS. To fairly evaluate the core performance against other reported FPGAs, a normalized performance metric of speed-up per logic cell/process technology was used. Based on the analysis, the core gained a speed-up of 1.34x when run with a fold of six as compared to the state-of-the-art. For higher performance, the core can be accelerated on higher density FPGAs such as on the XC72000T FPGA which is expected to generate up to 1000 PE systolic arrays.

Chapter 6
Design and FPGA Implementation of the Gapped BLAST with the Two-hit Method

This chapter presents the design and implementation of the Basic Local Alignment Search Tool (BLAST) with the two-hit method in hardware. The background of heuristic-based sequence alignment algorithm is described and then, relevant work on the implementations of such algorithms using various computation platforms is discussed. Following this, details of the design and implementation of the BLAST with the two-hit method on the Virtex-5 XC5VLX110 FPGA are presented. The implementation results are then presented alongside a comparison with the state-of-the-art. Finally, conclusions are drawn and plans for future work are described.

6.1 Introduction

Using dynamic programming-based sequence alignment algorithms to search for a query sequence i.e. a newly discovered biological sequence against huge biological databases that consist of millions of subject sequences is a time-consuming task in molecular biology. This is due to the intensive search strategy used by the algorithms such as the Smith-Waterman and the Needleman-Wunsch. This result in very long processing times when run on a standard desktop computer and is computationally expensive when computer clusters or high performance supercomputers are used. With the ever-increasing size of biological databases over the years, considerable effort has been devoted in reducing both computational complexity and operational costs. This includes the use of heuristic methods such as the FASTA or Fast Alignment and the BLAST or Basic Local Alignment Search Tool to speed up the search for biologically related sequences in the database. FASTA was developed by Lipman and Person in 1985 [16] and then the local alignment algorithm was further improved three years later [111]. The need for increased search speeds with existing algorithms led to the development of a better algorithm known as BLAST introduced by Altschul et al. in 1990 [17]. Although these heuristic algorithms are unable to produce results as accurate as the gold standard

Smith-Waterman algorithm, this method is widely-used in sequence alignment due to the prohibitive costs of optimal solutions in achieving results in a realistic time, especially given the exponential growth in database sizes. The BLAST algorithm searches for high scoring pairs of aligned words and uses the most meaningful pairs for alignment. Unlike with optimal alignment which searches for the optimal path between two sequences by calculating the entire alignment matrix, the heuristic algorithm calculates alignment only at regions with high scoring pairs. Other regions that are far away from the main aligned region in the matrix are discarded. This reduces the time taken to calculate the alignment matrix due to the lower search sensitivity. In 1997, Altschul et al. further improved the BLAST algorithm in terms of its speed and sensitivity. Thus, BLAST is even now still being used by biologists as an alternative to the optimal solution. Several refinements of the original BLAST algorithm include the use of the 'two-hit' method for extending hits instead of using multiple hits as in the original, and the use of gapped alignment for the final stage of BLAST in order to further improve its search sensitivity. BLAST has become the *de facto* standard in heuristic-based sequence alignment, and on-going tasks of its development and maintenance have been undertaken by the National Center for Biotechnology Information (NCBI). There are several variations of the BLAST algorithm depends on the types of input for queries and the database searched, as summarized in Table 6.1.

Table 6.1 : Variations of BLAST algorithm [112]

BLAST variations	**Input**	
	Query Types	**Database Type**
BLASTp	Amino Acid	Amino Acid
BLASTn	Nucleotide	Nucleotide
BLASTx	Translated Nucleotide	Amino Acid
TBLASTn	Amino Acid	Translated Nucleotide
TBLASTx	Translated Nucleotide	Translated Nucleotide

The following section discusses a more details background of this algorithm. Then, prior work regarding the acceleration of the BLAST algorithm in hardware is discussed in section 6.3.

6.2 Background

The BLAST algorithm aims to search for sequence homology in a minimized search space to get results in a realistic time with its search sensitivity approximate to the optimal ones. The newer version of BLAST i.e. BLAST with the two-hit method has significantly improved the accuracy of searched results at substantially augmented speed. Essentially, the BLAST with the two-hit method consists of three stages; namely seed generation, ungapped extension and gapped extension, as shown in Figure 6.1. The first stage comprises of word matching and two-hit finder operations. Before the word matching stage, a query sequence is pre-processed into a list of overlapping words of size *W*, also known as *W-mers*. Each of the words generated is then searched against a subject sequence in the database with the aim to search for highly similar pairs of pre-processed words against fragments of the subject sequence. A score is associated with each of the word matching operations and those matched words with scores higher than a predefined threshold value, *T*, are referred to as hits.

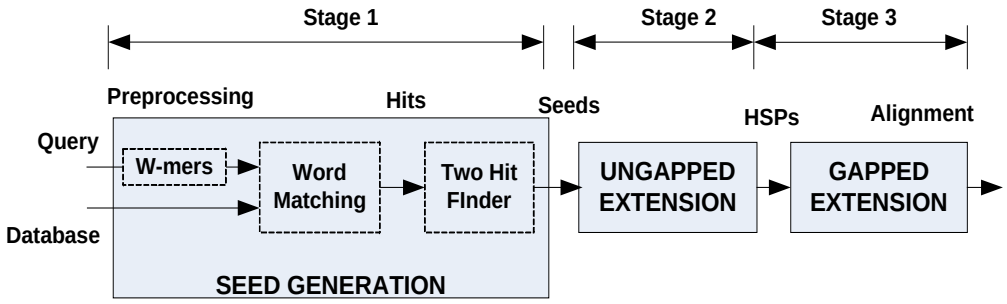


Figure 6.1: Overview of the stages in NCBI gapped BLAST with two-hit method

For the new generation of BLAST, a threshold parameter T is typically set at a lower value (i.e. 11) than that in the original BLAST in order to improve its search sensitivity. The two-hit finder unit filters out random hits and leaves only meaningful hits, which are also known as seeds, for alignment. The second stage extends the seeds by aligning them without allowing any gaps. Any pair with an ungapped alignment score exceeding

another predefined threshold value is then referred to as an HSP or high scoring pair and is used for alignment during the final stage. Unlike the original BLAST algorithm, which finds several alignments for a particular sequence in a database, the new BLAST has the ability to generate gapped alignments during the last stage. This alignment uses a dynamic programming approach to extend alignment from a central point of a high scoring pair (HSP) in both directions, to the left and right from the central point of the HSP. A detailed explanation of the stages involved in BLAST with the two-hit method is discussed in the following sections.

6.2.1 Seed generation

In this stage, a query sequence is pre-processed to develop overlapping sub-residues known as *W-mers* of size *W*, where *W* is dictated by a fixed residue length of sub-residue. This depends on the types of BLAST algorithm used; for example, for BLASTp, *W*=3 and for BLASTn, *W*=11. Note that, BLASTp is for protein sequences, while BLASTn is for DNA sequences. Figure 6.2 illustrates the pre-processing methodology of a query sequence *'H' 'E' 'A' 'G' 'A' 'W' 'G' 'H' 'E' 'E'* in the case of BLASTp. The number of *W-mers* or words in the list can be determined by *(q-W) +1*, where *q* = length of the query sequence. In this example, *q* = 10 and *W*=3. Then the total number of words generated is eight, as illustrated in Figure 6.2.

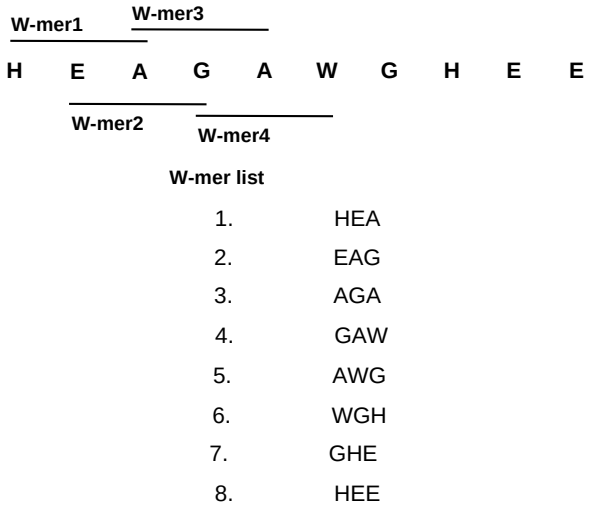


Figure 6.2: Pre-processing of a query sequence into a list of *W-mer* for BLASTp

An example of the word matching operation is illustrated in Figure 6.3. During this stage, pairwise sequence segments of the words are scored against subject sequences in the database using a score matrix such as the BLOSUM 62 matrix as shown in Figure 6.4.

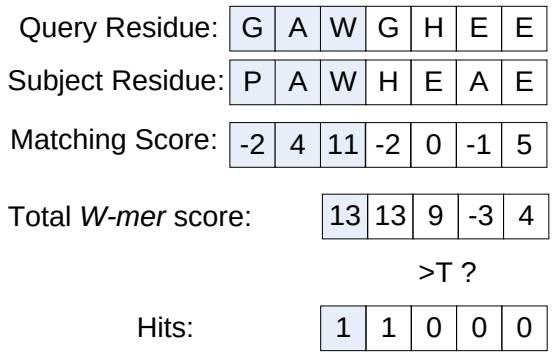


Figure 6.3 : Example of hit finding process with *W=3* and *T =11*. The score matrix used in this example is BLOSUM 62.

The BLAST 62 is the default substitution matrix in the gapped BLAST with the two-hit method [17]. Any word pair with a score satisfying a given threshold, *T* (*T*=11 for NCBI BLASTp) will be recorded as a hit.

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	4	−1	−2	−2	0	−1	−1	0	−2	−1	−1	−1	−1	−2	−1	1	0	−3	−2	0
R	−1	5	0	−2	−3	1	0	−2	0	−3	−2	2	−1	−3	−2	−1	−1	−3	−2	−3
N	−2	0	6	1	−3	0	0	0	1	−3	−3	0	−2	−3	−2	1	0	−4	−2	−3
D	−2	−2	1	6	−3	0	2	−1	−1	−3	−4	−1	−3	−3	−1	0	−1	−4	−3	−3
C	0	−3	−3	−3	9	−3	−4	−3	−3	−1	−1	−3	−1	−2	−3	−1	−1	−2	−2	−1
Q	−1	1	0	0	−3	5	2	−2	0	−3	−2	1	0	−3	−1	0	−1	−2	−1	−2
E	−1	0	0	2	−4	2	5	−2	0	−3	−3	1	−2	−3	−1	0	−1	−3	−2	−2
G	0	−2	0	−1	−3	−2	−2	6	−2	−4	−4	−2	−3	−3	−2	0	−2	−2	−3	−3
H	−2	0	1	−1	−3	0	0	−2	8	−3	−3	−1	−2	−1	−2	−1	−2	−2	2	−3
I	−1	−3	−3	−3	−1	−3	−3	−4	−3	4	2	−3	1	0	−3	−2	−1	−3	−1	3
L	−1	−2	−3	−4	−1	−2	−3	−4	−3	2	4	−2	2	0	−3	−2	−1	−2	−1	1
K	−1	2	0	−1	−3	1	1	−2	−1	−3	−2	5	−1	−3	−1	0	−1	−3	−2	−2
M	−1	−1	−2	−3	−1	0	−2	−3	−2	1	2	−1	5	0	−2	−1	−1	−1	−1	1
F	−2	−3	−3	−3	−2	−3	−3	−3	−1	0	0	−3	0	6	−4	−2	−2	1	3	−1
P	−1	−2	−2	−1	−3	−1	−1	−2	−2	−3	−3	−1	−2	−4	7	−1	−1	−4	−3	−2
S	1	−1	1	0	−1	0	0	0	−1	−2	−2	0	−1	−2	−1	4	1	−3	−2	−2
T	0	−1	0	−1	−1	−1	−1	−2	−2	−1	−1	−1	−1	−2	−1	1	5	−2	−2	0
W	−3	−3	−4	−4	−2	−2	−3	−2	−2	−3	−2	−3	−1	1	−4	−3	−2	11	2	−3
Y	−2	−2	−2	−3	−2	−1	−2	−3	2	−1	−1	−2	−1	3	−3	−2	−2	2	7	−1
V	0	−3	−3	−3	−1	−2	−2	−3	−3	3	1	−2	1	−1	−2	−2	0	−3	−1	4

Figure 6.4: The BLOSUM62 matrix

The next stage is to filter out unrelated hits using the two-hit finder. In this stage, only non-overlapping hits with a distance *A* (typically 40) from one hit to another that lies on the same diagonal will be selected for the ungapped extension stage. This stage differentiates between meaningful and random hits generated from the word-matching stage.

6.2.2 Ungapped extension

During this stage, each two-hit pair that was recorded in the previous stage is extended. Figure 6.5 illustrates an example of the ungapped extension operation of a two-hit pair. The solid rectangles represent a pair of significant words between given Query and Subject sequences, while the dashed rectangle marks the extension. The extension starts inward (arrow 1) from left to right to close the gaps and then proceeds outward to the ends (arrows 2 and 3). As in the seed generation stage, a score matrix is used to award scores for each residue pair of the two sequences. Unlike the seed generation stage presented in section 6.2.1, which only accumulates a score for each word of residues of size *W*, in this stage the score beginning from the start of the extension is accumulated until it drops to more than *X* below the maximum score so far. When the extension is finished, if the accumulated score exceeds another predefined threshold value, the two-hit pair is called a high-scoring pair (HSP) and subsequently used in gapped extension.

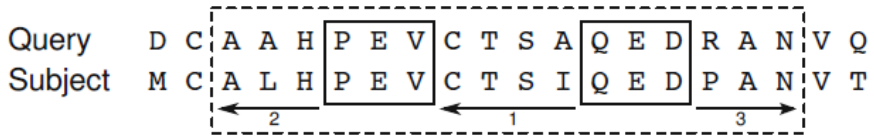


Figure 6.5: Example of an ungapped extension of a two-hit in the NCBI BLAST implementation [113]

6.2.3 Gapped extension

The gapped extension operation uses a modified version of the dynamic programming (DP) algorithms such as the Needleman-Wunsch and the Smith-Waterman to perform alignment matrix computation only in the regions of interest. In order to illustrate the operation of this heuristic-based sequence alignment search methodology, two biological

sequences; the *broad bean leghemoglobin I* and *horse β-globin* are aligned using the BLAST with two-hit method as illustrated in Figure 6.6. In this gapped extension stage, the alignment drives from the central point of the high scoring pair (HSP) towards both ends. The HSP which is located at the center of the diagram was generated by the inward ungapped extension as described in section 6.2.2. Recall that, optimal alignment approach as discussed in Chapter 4, calculates alignment matrix scores of the two biological sequences entirely. In heuristic approach, scores of the alignment matrix are calculated only based on regions of interest, typically, along the diagonal region of the alignment matrix as shown in Figure 6.6. The *X*-drop mechanism reduces the running time needed to compute the gapped extension and creates a region of the path graph towards the left and the right apart from the HSP as denoted by the black regions in Figure 6.6. The grey region shows that no alignment scores are calculated in this area as a result of the modification done to the alignment algorithm, whereby the alignment matrix computation is limited to a threshold value dictated by the *X*-drop mechanism.

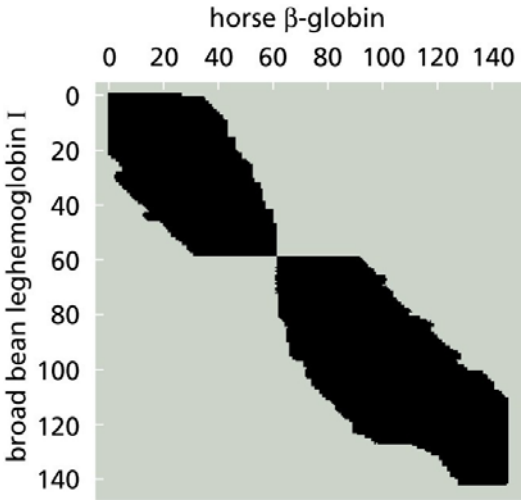


Figure 6.6 : Example of gapped extension of two biological sequences [114]

The following section first discusses the DP-based Needleman-Wunsch algorithm with affine gap penalty. Then, the modified version of this algorithm to limit the search to only on the regions of interest is discussed. The Needleman-Wunsch algorithm is chosen here as an alignment algorithm because it is the NCBI BLAST default algorithm in the gapped extension stage of the BLAST with the two-hit method.

6.2.3.1 The Needleman-Wunsch algorithm

The Needleman-Wunsch algorithm with linear gap penalty as shown in equation 6.1 was introduced by Needleman and Wunsch in 1970[13]. Given a query sequence $x = (x_1, x_2, x_3 \ldots x_n)$ of length n residues and a subject sequence $y = (y_1, y_2, y_3 \ldots y_m)$ of length m residues, a matrix $F: \{1,2,3, \ldots n\} x \{1,2,3, \ldots m\}$ is computed recursively for each $F(i,j)$ element in the alignment matrix (see Figure 6.7) taking into account data dependencies from the left, i.e. $F(i-1,j)$, the diagonal, i.e. $F(i-1,j-1)$, and the top, i.e. $F(i,j-1)$ for residues x_i and y_j. The $F(i, j)$ is the best score of the alignment up to prefixes x_i and y_j.

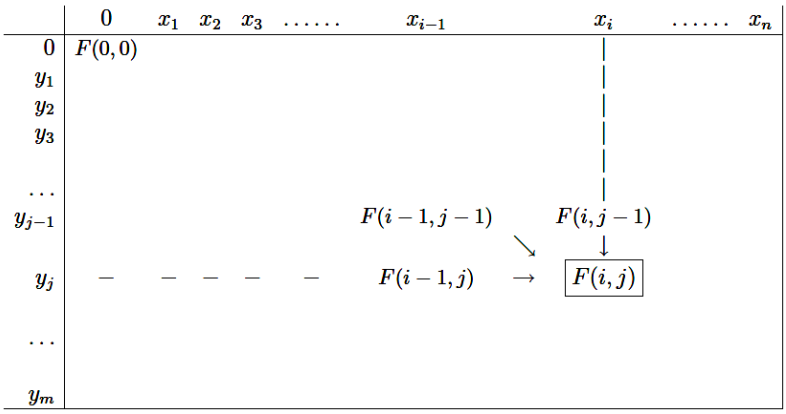


Figure 6.7: Illustration of the dynamic programming to compute best score $F(i,j)$ of two prefixes ($x_1, x_2, x_3 \ldots x_n$) and ($y_1, y_2, y_3 \ldots y_m$)

The recursive computation starts from $F(0,0)$ until $F(n,m)$ to search for the optimal alignment of the sequences x, and y. The time complexity of such an alignment is $O(mn)$.

$$F(i, j) = \max \begin{cases} F(i-1, j-1) + s(x_i, y_j) \\ F(i-1, j) - d \\ F(i, j-1) - d \end{cases} \qquad (6.1)$$

A more accurate model of the dynamic programming algorithm was proposed by Gotoh [15] in 1982. This is known as global alignment with an affine gap penalty. Unlike the linear gap penalty (equation 6.1) which penalizes gaps in alignment linearly, the gap penalty for the overall sequence depends on the affine gap function (equation 6.3), as described in section 2.3.2. This gap penalty is defined by two constants, gap open, d, and

gap extend, *e*, with *g* as the gap length. Alignment with the affine gap penalty reflects residue insertions and deletions, and hence is a more realistic model for biological phenomena [3].

$$penalty\,(g) = -gd \qquad (\ 6.2\ )$$

$$penalty\,(g) = -d - (g-1)\,e \qquad (\ 6.3\ )$$

However, this model requires three values; best score *F(i,j),* insertion from the *x* direction $I_x(i,j)$ and insertion from the *y* direction $I_y(i,j)$. The recursive equations 6.4, 6.5 and 6.6 of the algorithm are shown below.

$$F(i,j) = \max\begin{cases} F(i-1,\,j-1) + s(x_i, y_j) \\ I_x(i-1,\,j-1) + s(x_i, y_j) \\ I_y(i-1,\,j-1) + s(x_i, y_j) \end{cases} \qquad (\ 6.4\ )$$

$$I_x(i,j) = \max\begin{cases} F(i-1,\,j) - d \\ I_x(i-1,\,j) - e \end{cases} \qquad (\ 6.5\ )$$

$$I_y(i,j) = \max\begin{cases} F(i,\,j-1) - d \\ I_y(i,\,j-1) - e \end{cases} \qquad (\ 6.6\ )$$

6.2.3.2 The modified Needleman-Wunsch algorithm for gapped extension

For the gapped BLAST, some modifications are required. The gapped extension using the dynamic programming algorithm with the affine gap penalty is bounded to a certain limit, whereby the computation of scores terminates if the current score falls below a certain cut-off value *X* compared to it previously highest computed score. This reduces the time spent on computing for unrelated regions located far away from the alignment region. Furthermore, the traceback procedure may start from any location in the alignment matrix instead of starting from the bottom right corner as in the original algorithm.

6.3 Prior work on hardware implementations of the gapped BLAST with the two-hit method

Over the last decade, due to the exponential increase of biological databases, the BLAST algorithm has been accelerated on various computing platforms, including in high performance supercomputers, multiprocessor clusters and GPUs. The BlueGene [115] and the IBM Blade Cluster [116] with a total of 128 Intel Xeon 2.8-3.0 GHz processors are among the examples of high performance supercomputers and multiprocessor clusters used to accelerate heuristic-based sequence alignment. Although BLAST was scalable for higher performance according to the number of clusters, the maintenance, energy, costs and size associated with the accelerator were comparable to those of single-node solutions[117]. Alternatively, CUDA-BLASTp, with a GPU-based computing platform, has been reported [118] to have a speed-up of up to 10 on an NVIDIA GeForce GTX 280 graphic card compared to the sequential NCBI BLASTp version 2.2.22 which ran on the GeForce GTX 295. On the other hand, the BLAST algorithm has also been accelerated in hardware i.e. FPGAs. In one of the earlier implementations of the BLAST algorithm on FPGA[119], the BLASTn algorithm which is the version that searches for DNA sequences on the Mercury system. The latter was made up of an FPGA and a traditional processor with disk-based computation to support high data rates for computation. However, only stage 1 was implemented in hardware, while the other two stages were executed by the processor. Analysis on the BLASTn pipeline in [119] showed a significant reduction in data to be processed before the gapped extension stage, and hence the first stage was identified as one of the bottlenecks for the BLAST performance which needed acceleration. Further work was then carried out in 2007 by Jacob et al.[117]. The BLASTp was executed on the same platform, and seed generation, ungapped extension and a gapped extension pre-filter were implemented on two different FPGAs. Due to the limited number of blocks RAM in the FPGA, the implementation used two FPGAs to combine all of the stages aiming for higher performance. Both stages 1 and 2 were implemented on the XC2V6000 FPGA and the gapped extension pre-filter stage was implemented on the XC2V4000 FPGA resulting in a speed-up of 37x compared to the NCBI BLASTp version 2.2.10 which ran on a single 2.8 GHz Pentium 4 workstation with 1 GB memory. The gapped extension was executed by a host CPU. In the same year, Sotiriades et al. presented a general

reconfigurable architecture for all BLAST algorithms (i.e. BLASTn, BLASTp, BLASTx, tBLASTn and tBLASTx) which was implemented on the XC4VFX140 FF1517-11 FPGA with a reported speed-up of about 5.85 faster than on the Mercury BLAST system. However, no gapped BLAST or two-hit method was implemented on the core, thus it lacked of superior advantages of the newer version of the BLAST algorithm. In 2010, Mahram and Herbordt [120] from the Boston University used another approach by utilizing a single Altera Stratix-III FPGA connected to 4.5 GB DRAM to implement three different filters on FPGA i.e. the two-hit finder, the ungapped alignment and gapped alignment filters. The reported performance was 25 to 30x speed-up compared to the NCBI BLASTp version 2.2.20 which ran on the 64-bit 3 GHz Xeon Quad processor (Harpertown X5412) with 8 GB of memory. Recently, L.Wienbrandt et al. in [113] accelerated the BLASTp on multiple FPGAs using the RIYERA FPGA-based hardware platform. RIYERA was first introduced as COPACOBANA 5000 for bioinformatics applications [121]. It comprises of 128 low cost Spartan3-5000 FPGAs to enable massive parallel computation with reconfigurable capabilities. The reported speed-up compared to the one-thread NCBI BLASTp v.2.2.25+ using a fully utilized 2x Intel Xeon E5520 PC system operated at 2.26 GHz was up to 376x. The BLASTp pipeline was closely similar to that in the work reported in another study, [122], whereby query pre-processing was executed by the host.

The seed generation stage accounts for 50 percent of the execution time [119], [117] of the BLAST operation. Thus, acceleration of the BLAST stage was then implemented in hardware as reported in [119], [117], [120], and most recently in [113]. To achieve higher performance, all stages of the BLAST have to be accelerated in hardware. This has prompted to the need for the fully parallel implementation of all BLAST stages. However, implementing all three stages in hardware requires a significant amount of BRAM, especially to store a score matrix such as BLOSUM 62 inside the PE pipeline at each stage. Therefore, in this research, new hardware architecture is presented which fully-pipelined all of the BLAST stages. In addition, a well-known double buffering technique is adopted in the hardware design to hide the configuration time of the configuration elements (CEs) in each stage of computation. In the following section, the corresponding hardware design and implementation of the BLASTp with the two-hit method are discussed.

6.4 Our FPGA-based implementation of gapped BLAST with the two-hit method

In Figure 6.8, all of the BLAST stages mentioned in section 6.2, including the query pre-processing step, are implemented in hardware. The Hit Finder, Ungapped Extender and Gapped Extender are made up of pipelines of processing elements (PEs). Each PE holds one query residue at a time for processing.

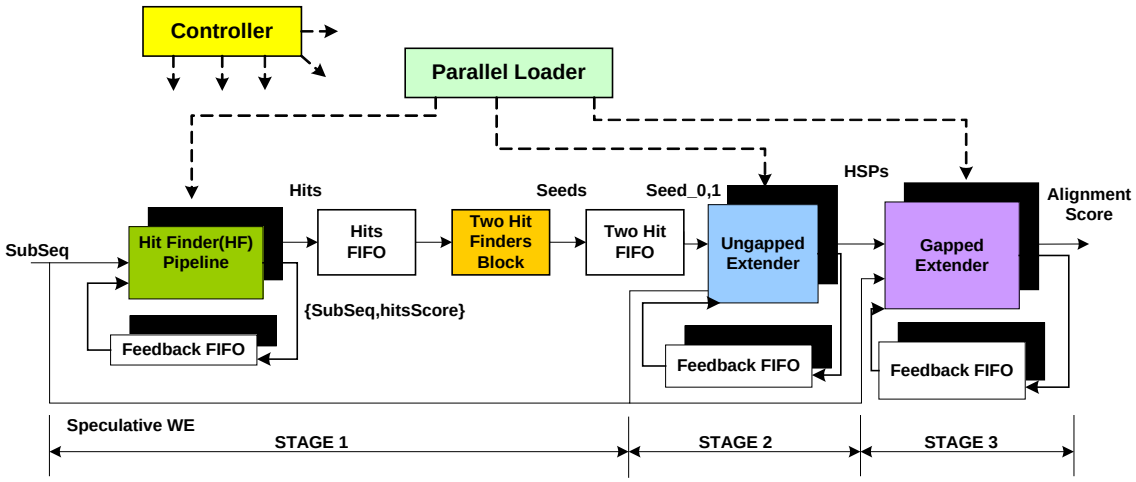


Figure 6.8: A fully-pipelined BLASTp hardware architecture

In reality, biological sequences are often hundreds if not thousands, long. Therefore, in the cases of query length longer than the number of PEs physically implementable on the FPGA device in hand, the core supports multiple-pass processing. An efficient scheduling strategy based on the double buffering technique is adopted in the core's architecture to perform multiple-pass processing with negligible PE configuration time. Moreover, the limited block RAM resources, which is the performance bottleneck of the FPGA-based BLAST implementations reported in literature is addressed by having a parallel loader to simultaneously configure processing elements.

6.4.1 PE with double buffering CEs

A novel PE architecture with two configuration elements (CEs) as shown in Figure 6.9(a), is designed with aim to reduce the PE dependency on the restricted block RAM resources in the FPGA. This way, all the BLAST stages could be accelerated in hardware with parameterizable PEs in order to achieve higher performance. Typical

FPGA implementations use block RAM to store a score matrix such as the BLOSUM62 per PE. Consequently, PE replications at all BLAST stages consume a considerable proportion of the block RAM, which limits the core's performance due to its restricted block RAM resources. In the new PE architecture, the CE is made up of the FPGA's abundant logic slices and only two configuration elements are used alternately during multiple-pass computation. At each BLAST stage, the PE requires a copy of the score matrix to perform its processing task, whereby the score matrix is used to score residue pairs of a query and a subject sequence. Since only two CEs are allocated per PE, then a PE can hold two columns of the score matrix at a time. Therefore, the double buffering technique, as stated earlier in this chapter, is adopted in the core's architecture to enable PE processing without interruption. This is done by an efficient strategy for scheduling between CE configuration and the PE processing operation, whereby while one CE is used for PE processing, the other CE is updated with a new column of the score matrix for subsequent pass computation and vice versa. Figure 6.9(b) illustrates such overlapping operations. In this scheduling strategy, the CE configuration time is virtually removed during multiple-pass computation, except for the initial configuration time of CE_0, which is negligible compared to overall computation time. A folding factor of four is assumed, where the length of a query sequence is 4 n_{PE} residues and only n_{PE} could be implemented in hardware. Therefore, computation for the entire length of the query sequence proceeds in four passes: F_0, F_1, F_2 and F_3. To enable the efficient scheduling strategy, both processing and CE configuration are set to occur simultaneously, whereby during the first CE holds a column of a score matrix for computation, the other CE is configured with the subsequent score matrix column and vice versa in the subsequent pass. In this architecture, this scheduling strategy is referred to as overlapped computation and configuration (OCC). As a result of the overlapping operations, the overhead time to configure the CE is virtually removed. In addition, the PE architecture also successfully reduces its dependency on the restricted memory resources in storing a score matrix for computation.

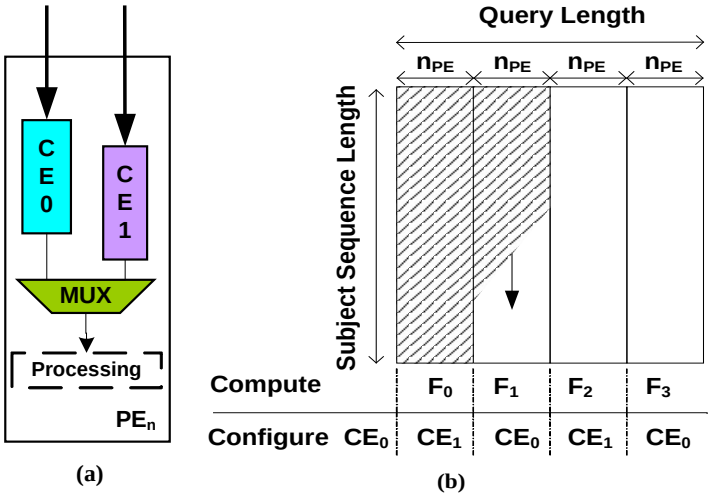


Figure 6.9: (a) Internal PE structure with fixed configuration elements (CEs)
(b) Computation and configuration over the same systolic array

Figure **6.10** illustrates the mapping between the CE and its corresponding query sequence residue. In this example, a query sequence "*P*, *A*, *W*, *G*, *H*, *E*, *A*, *E*" of a length of eight residues and the maximum number of the PE hit finder of four PEs is assumed for simplicity of explanation.

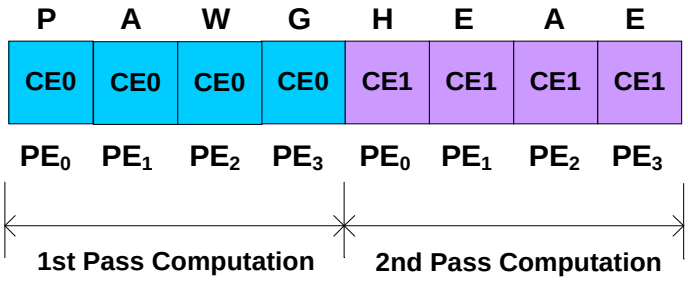


Figure 6.10 : The query residue to CE mapping in the case of two-pass computation

Each query residue is mapped to its allocated CE, and the first four residues of the query sequence are allocated for CE_0 with the last four residues for $CE_{1.}$ The allocation between the CE and its corresponding score matrix column is dictated by the query residue held by the CE. For example, the CE_0 in PE_1 is allocated with the query residue '*A*'. During CE configuration, all elements in column '*A*' of a given score matrix such as the BLOSUM62 are stored in that CE. Similarly for all CEs, the query residue held by the CE dictates which score matrix column is stored in its corresponding CE. To enable a

smooth transition between one computation pass to another, CE configuration occurs simultaneously.

6.4.2 Seed generator

The seed generator performs three different tasks; query pre-processing, word matching, and two-hit finding, as outlined in section 6.2.

6.4.2.1 Query pre-processing and word matching

In the query pre-processing task, instead of directly generating a list of *W*-mers from a given query sequence, the query sequence is stored in a linear systolic array as discussed in section 6.4.1. The array with a pipeline of Hit-Finder PEs performs both tasks and accumulates each *W*-mer score, SHF, following equation 6.7.

$$\sum_{i=0}^{W-1} S_{HF}(Q[i], S[j]) \geq T \qquad (6.7)$$

where *Q[i]* is the i^{th} residue of a query sequence, *S[j]* is the j^{th} residue of a subject sequence, *T* is a given threshold value, and *W* =3. The corresponding hardware implementation of the hit finder block is illustrated in Figure 6.11. The PE calculates the query-subject sequence's residue score, using a scoring matrix in the CE, and propagates the score to the subsequent PE. Any word pair with its accumulated score satisfying the condition in equation 6.7 is recorded in the PE Hits FIFO in the form of the address of its corresponding query and subject sequence residues. Any accumulated scores which are less than the predefined threshold are discarded. The word length of the FIFO is parameterizable to allocate the addresses of both the query and subject sequence residues. As the hit finder block finishes calculating the hit scores of a subject sequence, the PE hit FIFO shifts the recorded hits through a serial chain as shown by the bold dotted line in Figure 6.11, to another FIFO (the Hits FIFO) outside the PE hit Finder block. Following this, a new subject sequence is fetched for another hit finding operation. The hit finding process continues until all subject sequences in the database are exhausted. The hits in the FIFO include random hits and meaningful hits for

alignment. Therefore, to filter out the unrelated hits, a Two-hit Finder unit is used, and details of its implementation are discussed in the following section.

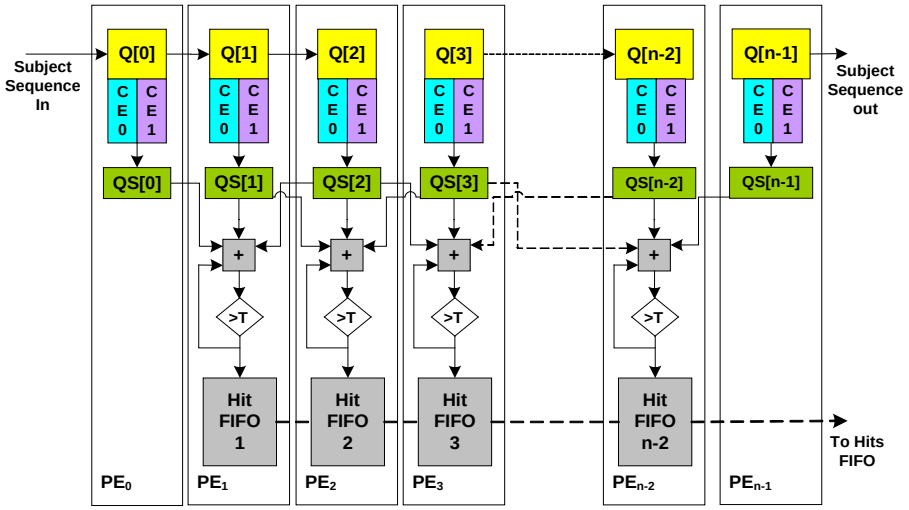


Figure 6.11 : Internal structure of the hit finder block

6.4.2.2 The two-hit finder

The two-hit finder searches for any two non-overlapping hits with a distance less than A (typically 40) and more than W between one to another. The filtered hits must also be located on the same diagonal. Any hit pair satisfying these conditions is selected for ungapped extension. These two hits are referred to as seeds. In hardware, the hit finder is implemented by a logic unit as illustrated in Figure 6.12. The numbers of two-hit finder units are parameterizable and each of them runs in parallel to reduce the time spent to search for seeds. Each of the two-hit finders comprises of two subtractors; one calculates the distance between the two hits in the x-direction, which is the difference between two query addresses, Δx, and the other calculates the distance between the two hits in the y-direction which is the difference between two subject sequence addresses, Δy. If Δx is within a given threshold value, A, and if Δx equals Δy (so that the hit pair is on the same diagonal), then the addresses (residue locations from both query and subject sequences) of the two hits are stored in the two-hit FIFO for the subsequent stage of ungapped extension.

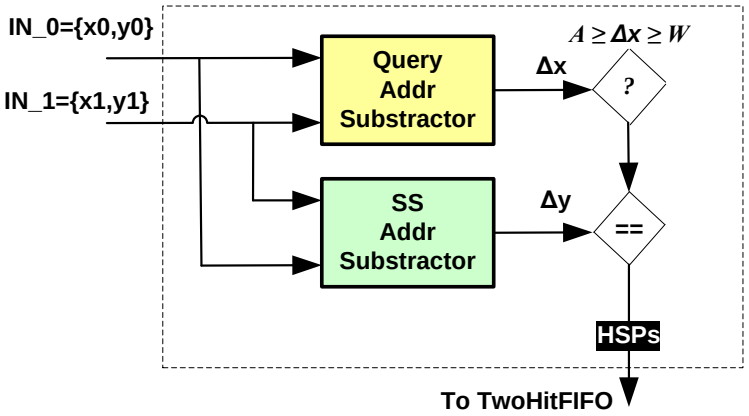


Figure 6.12: Simplified structure of the two-hit finder

6.4.3 Ungapped extender

The Ungapped Extender block implements the ungapped extension as discussed in section 6.2.2. It is implemented by a logic unit as shown in Figure 6.13 which comprises of two ungapped extender blocks. Each of the blocks is made up of a pipeline of PE arrays. The *Ungapped Extender_1* is used for inward extension and in the case of outward extension both *Ungapped Extender_0* and *Ungapped Extender_1* run in parallel to extend the alignment in both directions. Each seed that is stored in the two-hit FIFO is read into the ungapped extender in turn. The inward ungapped extension starts from one seed point (*Seed_1*) to the other seed point (*Seed_0*). As extension proceeds, the query-subject residue pairs are scored along the extension against a score matrix in the configuration element of the PE. Then, the ungapped extension proceeds outward towards their ends. Similarly, as the extension proceeds, the query-subject residue pairs along the extension are scored using the score matrix without allowing any gaps between alignments. Extensions in either direction terminate if the current accumulated score falls below the cut-off value *X* compared to the maximum score so far or if any of the extensions reaches the end of the query or subject sequence. The accumulated inward and outward scores are then summed, and if the total score exceeds a given threshold value, *X*, then the seeds (*Seed_0* and *Seed_1)* now known as an high scoring pair (HSP) are stored in the UE FIFO for gapped alignment.

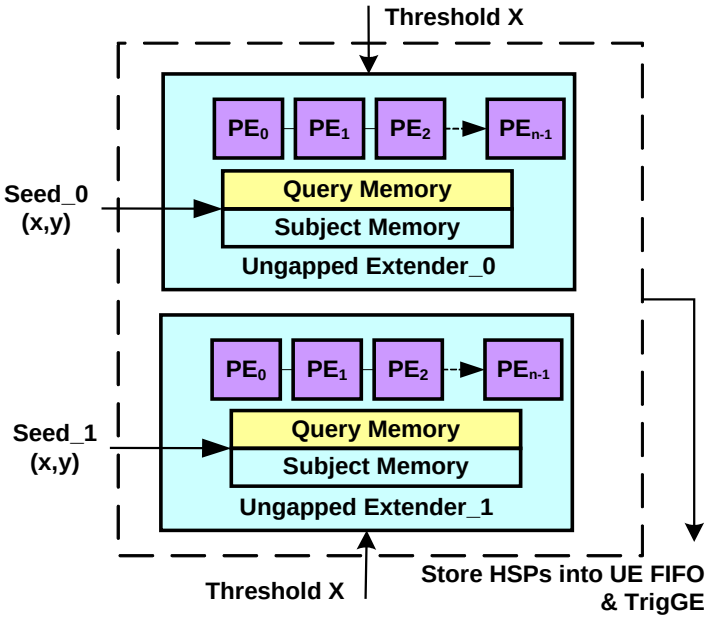


Figure 6.13 : The simplified inner structure of the ungapped extender block

Note that, before the abovementioned inward and outward extensions start, the locations of the query and subject sequences' residues are fetched based on the information concerning their locations provided by the seeds to initiate the starting points of the extension. The operation starts when the fetched query residue is read from the Query Memory and mapped to its corresponding CE in the PE. Once the CE configuration has finished, the subject sequence residues are fetched from the Subject Memory for the ungapped extension process. The query and subject sequences are pre-stored inside the memory and in the case where no seed is found for a particular subject sequence, a new subject sequence is written into the memory. The decision about whether to save or discard a particular subject sequence is based on the seeds generated by the two-hit finder in the seed generation stage described in section 6.4.2.2.

6.4.4 Gapped extender

The hardware architecture for the gapped extender is similar to the one presented in Figure 6.14 except in two respects; 1) the gapped alignment starts from the central pair of the inward alignment calculated by the ungapped extender and the extension proceeds outwards in both directions. 2) The internal PE architecture is slightly more complicated

than the one that is implemented for the hit finder and ungapped extender blocks. This is because the PE with gapped extension implements the elementary calculation of a modified version of the dynamic programming algorithms for gapped alignment, as discussed in section 6.2.3.2. The runtime for the gapped extender is reduced as a result of the *X*-drop mechanism, whereby no further extension occurs if the extension score falls a value of *X* below the maximum score so far. In this architecture, the final alignment is not implemented in hardware due to the excessive memory required to store the trace back pointers of the alignment, and therefore only the best score for each subject sequence is calculated.

6.4.5 The controller with the efficient scheduling strategy

The efficient scheduling strategy as presented in section 6.4.1 is implemented in the controller's states machine in the BLASTp core architecture in order to support multiple-pass computation. Such an operation occurs in the PE at all BLAST stages. The scheduling strategy manages the fixed configuration elements (i.e. two CEs) efficiently by alternately using each of them for PE processing and CE configuration. To enable independent operations, each of the BLAST stages is controlled by a separate controller. This allows for the seed generation stage, which is the most time-consuming operation in any of the BLAST stages, to run independently and the search for hits occurs without interruption as long as the Hits FIFO is not full of hits. The main controller manages seed generation and synchronizes its operation with the other two controllers. The sub-controllers control the ungapped and gapped extensions, including CE configuration and PE computation during alignment. These two stages run in parallel with seed generation in order to achieve maximum parallelization. Synchronization between the three is crucial to enable an anti-stall mechanism between stages. This is implemented by controlling the incoming subject sequence from the database, which is stored in the host's memory. If the hits FIFO in the first stage and the two-hit FIFO in the second stage are not full, then a new subject sequence is fetched into the hit finder block for hits processing, otherwise, no new subject sequence is fetched for hit finding.

6.5 Implementation results

The BLASTp core architecture has been captured using Verilog HDL in a parameterizable manner. The designed core is implemented on the Alpha Data board with Xilinx Virtex-5 FPGA on it. The XC5VLX110-3FF1153 device offers a total of 17, 280 logic slices or 110,592 logic cells and 256 blocks RAM of size 18Kbit each. In terms of hardware resources utilization, the 65 nm FPGA fits a maximum PEs of 100, 50 and 50 for stage 1, stage 2 and stage 3 of the BLAST algorithm respectively. Details of the resources utilization for the BLAST stages are summarized in Table 6.2

Table 6.2 : Hardware resource utilization of a PE in each BLASTp stage. All resources are extracted from the Xilinx ISE 13.2 place and route report.

BLASTp Stages	**Stage 1**	**Stage 2**	**Stage 3**
Performance Parameters	**PE Hit Finder**	**PE Ungapped Extender**	**PE Gapped Extender**
Clock Frequency (MHz)	245.3	247.6	221.0
Computation word length (bit)	11	11	11
Configuration word length (bit)	5	5	5
Slices	71	56	118
LUTs	226	92	426
FFs	78	27	430
Block RAM (18k)	1	0	0

Each of the BLAST stages supports a folded systolic array architecture, whereby multiple-pass computation with the efficient scheduling strategy can be used if the length of the query sequence is longer than the physically implementable number of PEs. In terms of the complexity of the internal processing element circuit, the internal architectures of both the hit finder and the ungapped extender PEs are slightly simpler than that of the gapped extender PE. This is because the PE in the gapped extender stage requires additional computational units to implement the modified version of the dynamic programming algorithm with the affine gap penalty function outlined in section 6.2.3.2.

In terms of throughput performance, speed-up is used to evaluate the performance of the proposed core against the corresponding software implementation, i.e. the NCBI BLASTp, as well as comparing it with other FPGA implementations. The speed-up is calculated by dividing the execution time of the NCBI BLASTp by the execution time of the proposed core, as expressed in equation 6.8.

$$speed\ up = \frac{t_{BLASTp}}{t_{FPGA}} \qquad (6.8)$$

where t_{BLASTp} = the total execution time of the BLASTp software
t_{FPGA} = the total execution time of the designed core in FPGA

In the case of comparison against the software implementation, a set of different query sequences was extracted from the UniprotKB/Swiss-Prot protein knowledgebase version 2012 [8]. The length of query sequences ranges from 100 up to 2048 residues. The database sequences were also extracted from the UniprotKB/Swiss-Prot protein knowledgebase with 538,010 subject sequences and 190,998,508 protein residues. In this evaluation, biological sequences in the database are assumed to be already held in the accelerator card's memory because, in practice, sequence alignment is made against fairly static databases. For fair comparison performance of the BLASTp , the pure software implementation (i.e. the BLAST 2.2.27+) was executed on the Intel Dual Core Processor (E6600). This desktop computer platform is comparable to the FPGA device used in this comparison as both computation platforms have been fabricated from the 65nm process technology and they come from the midrange type of FPGA and processor respectively. The desktop computer is operated with an operating clock frequency of 2.0 GHz. For the hardware implementation, the designed core was clocked at 200 MHz with 100 PE Hit Finders. The corresponding implementation results of both platforms are summarized in Table 6.3.

Table 6.3: Speed-up performance of the proposed core against BLAST 2.2.27+. the proposed core was clocked at 200 MHz and search various lengths (100 to 2048 residues) of query sequence against a database sequence of 538,010 subject sequences and 190,998,508 protein residues

Length	Query Accession #	PEs	Fold Factor	H/W (s)	Software (s)	Speed Up
100	P02652	100	1	0.18	6.37	11.91
222	C5DTC6	111	2	0.75	10.73	14.31
246	P00762	82	3	1.24	12.82	10.33
376	P0C9N5	94	4	1.97	20.39	10.35
570	Q9LU36	95	6	2.95	69.52	23.61
800	B3KY11	100	8	3.92	56.34	14.37
1000	A8KA62	100	10	4.93	103.62	21.04
1600	D3DNT2	100	16	7.71	95.33	12.37
1800	Q9BYP7	100	18	8.98	121.29	13.51
2048	Q8IYD8	103	20	9.81	203.93	20.79

Based on the experimental work, the designed core with various folding factors (i.e. up to 20), achieved more than a tenfold average speed-up against the NCBI BLASTp. The multiple-pass processing (with fold factors as in Table 6.4) is required in this case, due to the limitation s of current hardware resources, whereby only 100 PEs are implementable in hardware. Since the design is not constrained by any particular FPGA device, and also supports a scalable number of PE systolic arrays, the designed core can be implemented on other higher density FPGAs including the XC6VLX760, XCV6SX475T and XC6VHX350T in order to achieve higher performance.

For comparison against other FPGA implementations, previous studies in [113], [117], [122] and [123] are used as reference. Due to the different datasets reported in the literature, an effective straight comparison against other FPGA implementations cannot be made. For instance, in [113] BLASTp was implemented on the Spartan XC3S500 FPGA with concurrent processing of several queries. Other factors include the use of a group of query sequences as in [113], [117], [123] and[124]. To fairly evaluate other

BLASTp implementations in hardware with the designed core, each of the reported cores would have to be implemented on the same FPGA device and tested using the same sets of query and database sequences. However, this is not possible due to having no access to the cores used in the reported FPGA implementations. Alternatively, the best reported runtimes presented in the literature are calculated taking into account different sizes of databases and lengths of query sequences used. Table 6.4 summarizes the core performance for single pass computation against other FPGA implementations. It is obvious that the proposed core achieves at least 3x speed-up performance against other FPGA implementations.

Table 6.4: Speed-up performance of the proposed core against other BLASTp implementation on various FPGAs. Selected query length of 100 residues, DB 538,010 sequence, 190,998,508 residues.

Reference		Year	Device	Freq (MHz)	Execution time (s)	Speed-Up
Bleris. et al.	[113]	2012	XC3S5000	100	3.70	20.56
Jacob et al.	[117]	2007	XC2V6000	15	0.64	3.56
Herbordt et al.	[124]	2006	XC4VLX160	100	2.15	11.94
Sotiriades et al.	[123]	2007	XC4VFX140	100	0.79	4.39
Kasap et al.	[122]	2008	XC4VLX160	20	7.89	43.83
Proposed		2012	XC5VLX110	200	0.18	1.00

In an effort to measure speed-up performance of the designed core independently of area and fabrication process technology used, the speed-up figures in Table 6.4 are normalized with respect to area and the FPGA's look-up table (LUT) delay. In terms of area utilization, the studies cited in Table 6.4 reported the PE utilization in the form of logic slices, whereas the internal CLB slice architecture varies depending on types and families of FPGAs. For instance, one Virtex-5 FPGA slice is equivalent to two slices of its predecessors beginning from Virtex-4. Each Virtex-5 slice has four logic cells (LCs), whereas earlier generations of Virtex FPGAs have only two LCs per slice [125]. Among other elements, an LC is made up of a LUT, a register and a multiplexer. The logic cell is an abstract logic resource that measures area utilization independently of slice

architecture in any particular FPGA family. Therefore, LC is used rather than number of slices as an area normalization factor in order to effectively evaluate speed-up performance across different types of FPGAs. Then, the total LCs used in the PE is calculated using the aforementioned LC-slice relationship.

In addition, the amount of block RAM used in the PE to store substitution matrix scores also has to be taken into consideration prior to the normalization. To take into account memory utilization in the form of LCs, both the Gapped Extender PE and the Feedback FIFO of the designed core are synthesized using Cadence Build Gates version 2005 with 0.18um UMC process technology and the gate counts of each unit is noted. Firstly, the equivalent gate counts-LCs relationship of the Gapped Extender PE is calculated. The total gate counts utilized by the PE is extracted from the Cadence Build Gates tool, while total utilization in terms of logic cells is determined by multiplying the reported slice utilization in the Xilinx ISE 13.1 place and route report by two (for Virtex-4 or older FPGAs) or four (for Virtex-5 and newer FPGAs). Based on these relationships between gate counts and LCs, it is noted that one LC is equivalent to 443 gates. Then, using the gate counts-LCs relationship, equivalent LCs/Kbit of the Feedback FIFO (memory utilization) is calculated by dividing the total number of gate counts of the Feedback FIFO by 443 gates. From this analysis, one Kbit of memory (block RAM) utilizes an equivalent of 8174 gates or 18 LCs. Ultimately, the established LC relationships between both the logic and the memory-based elements are used as a reference benchmark to effectively normalize speed-up performance taking into account both the logic and memory elements used in the form of logic cells.

The normalized speed-up performance of the respective FPGA implementations is summarized in Table 6.5. The area ratio is calculated by dividing the total LCs of the proposed core with the total LCs of each of the reported FPGA implementations. The core's performance per LC is then calculated by dividing the raw speed-up in Table 6.4 by the area ratio. In this analysis, the designed core achieved at least 7x speed-up normalized per area compared to others. To evaluate the core's performance independent of fabrication technology, the ratio of the LUT delay of the corresponding FPGA implementations is calculated. Prior to that, the LUT delay ratio of the respective FPGA devices used in this comparison is calculated by dividing the LUT delay of the Virtex-5 FPGA with LUT delay of each device. Then, the area normalized speed-up is multiplied by the LUT delay ratio to get the normalized speed-up per area per process technology.

Based on the normalized figures, the overall speed-up of the BLASTp core with fixed CEs achieved at least 1.7x against others. The normalized figure shows that the designed core outperformed others independent of area and process technology.

Table 6.5: Normalized speed-up performance per area and process technology of the proposed core against other FPGA implementations

Reference		**Total Area (#Logic Cells)**	**Area Ratio**	**LUT Delay Ratio**	**Speed-up/Area**	**Speed-up/Area/ Process technology**
Bleris. et al.	[113]	62,786	1.70	0.19	12.09	2.30
Jacob et al.	[117]	143,700	0.48	0.23	7.42	1.71
Herbordt et al.	[124]	200,724	0.53	0.53	22.53	11.94
Sotiriades et al.	[123]	305,184	0.35	0.53	12.54	6.65
Proposed		106,668	1.00	1.00	1.00	1.00

6.6 Summary and conclusions

This chapter has presented a detailed design and implementation of heuristic-based sequence alignment in hardware. The gapped BLASTp with the two-hit method core architecture has been captured using Verilog HDL with all the BLAST stages (seed generator, ungapped extender and gapped extender) pipelined together to benefit from the maximum parallelism offered by the Xilinx Virtex-5 FPGA. A fixed number of configuration elements (only two CEs) have been designed in the PE architecture to hold two different sets of substitution matrix coefficients at a time. This has successfully optimized the logic resources required for the CE to alternately hold such coefficients during multiple-pass computations of the folded systolic array architecture. To enable the scalability of the PE systolic arrays, the CE has been designed from the FPGA's abundant logic slices. This new architecture has successfully addressed the block RAM limitation found in most of the reported FPGA-based BLASTp implementations. In addition, a parallel loader has been designed into the core architecture to enable configuration of the CEs in the aforementioned BLAST stages within a bounded configuration time. This way, alignment matrix computation and CE configuration with

new substitution matrix scores in operation between each of the BLAST stages occurs seamlessly.

The implementation results have shown that the designed core achieved an average speed-up of 15x as compared to the BLAST 2.2.27+ 'software only' implementation which ran on a comparable desktop computer. In the case of comparison with other reported FPGA implementations, the normalized performance indicator (speed-up/logic cells/process technology) was used to effectively compare the proposed core against others. The results showed that, through the fully pipelined BLAST architecture, the proposed core achieved up to 11x speed-up. The results presented in this research work were based on multiple-pass computations and the performance degrades as the number of computation passes increased. This is due to the slow-down factor, which proportionally increased with the number of folds. Therefore, to achieve higher performance, the designed core can be redeployed onto larger and faster FPGAs with minimal design effort. An example of such FPGA devices includes the XC6VLX760 FPGA, which offers 7x higher logic density than the XC5VLX110 FPGA used to implement the gapped BLASTp with the two-hit method architecture.

Chapter 7
Evaluation of FPGAs as a High Performance Solution for Biological Sequence Alignment

This chapter evaluates the efficiency of FPGAs in terms of area, throughput, power and energy consumption in the context of biological sequence alignment. Then performances per energy used and per dollar spent are calculated to evaluate the suitability of FPGAs as a viable alternative platform as compared to GPUs and GPPs, particularly for biological sequence alignments.

7.1 Performance Efficiency: FPGA vs. GPU vs. GPP

This section starts by comparing the performance efficiency of the aforementioned computational platforms in aligning biological sequences. In the case of FPGA implementation, the experimental work and corresponding results are based on the proposed architectures, and details of the design and hardware implementation have been presented in Chapters 4, 5 and 6 of this thesis. In this analysis, the database in each computing platform is assumed to be already held on the accelerator card's memory. This assumption is reasonable since sequence alignment is performed against static databases. Thus, the execution times presented in this chapter do not include the database transfer time. The most time-consuming part of the sequence alignment algorithms is the matrix filling operation. Therefore, the widely-used acceleration platforms for sequence alignments such as FPGAs and GPUs focus on this time-consuming part to accelerate the search before final scores from each alignment are transferred to the host.

7.1.1 Area

In chapter 3, the area efficiency of the FPGA against other computing platforms including ASIC and GPP has been discussed. K.Tatas et al. in [36] reported that the FPGA comes second in terms of area efficiency as compared to ASIC. This is because the circuits and interconnections in ASIC-based application designs are customized and

tailored to specific needs. Although the area utilization of such special purpose chip is relatively small compared to the equivalent implementations in other computing platforms, ASIC is non-reprogrammable after fabrication. On the other hand, FPGA is a generic platform which offers a 'sea of gates' for implementing wide ranges of applications. Single FPGA chip can be used to realize different applications due to the reprogrammable logic and interconnections. The re-configurability and flexibility of FPGAs have made this semiconductor device reprogrammable by end users after manufacture. Compared to GPPs and GPUs, FPGAs have the smallest footprint and this trend is likely to continue due to advances in transistor fabrication technology. For instance, a single Virtex-7 chip occupies footprint as small as a human. The latest Xilinx FPGA has been fabricated with 28 nm process technology with a capacity of up to two million logic cells.

7.1.2 Throughput

Speed-up and cell update per second (CUPS) are the most widely-used performance parameters in computational biology. In this comparative study, speed-up is calculated using equation 7.1, where $t_{GPPorGPU}$ is the GPP or the GPU execution time and t_{FPGA} is the FPGA execution time. The ratio of the execution times represents the FPGA speed-up against the reference platform.

$$speed\ up_{FPGAorGPU} = \frac{t_{GPPorGPU}}{t_{FPGA}} \qquad (7.1)$$

On the other hand, the cell update per second (CUPS) performance is determined by equation 7.2, where Q_{lentgh} and D_{length} are the total number of residues of the query and database sequences respectively. The term t is the total execution time which elapses during the homology search of a query sequence against known sequences in the database.

$$GCUPS = \frac{|Q_{lentgh}| \times |D_{lentgh}|}{t \times 10^9} \qquad (7.2)$$

In this section, the three biological sequence alignment core architectures (the DP-based sequence alignment, profile HMM-based sequence alignment and heuristic-based sequence alignment) presented in Chapters 4, 5 and 6 of this thesis are used to evaluate

FPGA performance. In the case of GPU implementation, the best reported work in the literature is used as reference. On the other hand, the 'software only' implementations of the respective types of sequence alignments were executed on a comparable desktop computer platform, the details of which are described later.

The DP-based sequence alignment core was implemented on the XC5VLX110-3FF1153 FPGA with 150 MHz operating clock frequency. The alignment algorithm was set to local alignment with a gap open penalty of -10, a gap extend penalty of -2 and the BLOSUM52 was used for the amino acid probabilistic model. In the case of GPU implementation, the work reported in [126] is used as reference. In this reported GPU implementation, the NIVIDIA GeForce 9600GT was used to accelerate the DP-based algorithm by implementing multi-threaded parallel computation on the Compute Unified Device Architecture (CUDA) platform. CUDA is essentially a parallel programming model and computing platform developed by the NVIDIA Corporation[127]. The corresponding software execution of the Smith-Waterman algorithm i.e. the SSEARCH version 35.05, was executed on the 65nm Intel Dual Core processor (E6600) with an operating clock frequency of 2.0 GHz. The SSEARCH is an open source and mature software tool from the FASTA program for performing DNA or protein-based Smith-Waterman sequence alignment. The implementation results for the respective platforms are summarized in Table 7.1. The execution times shown in the second column are based on a query sequence length of 256, since this is close to the average sequence length in the UniProtKB database. The search was performed against the UniProtKB database release 15.12, with a total of subject sequences of 513,877 or a total of 188,625,310 residues. In terms of speed-up and CUPS performance, the proposed FPGA implementation was faster by two orders of magnitude than the GPP solution. The GPU comes second fastest after the FPGA with CUPS and speed-up performance of one order of magnitude higher compared to the GPP solution.

Table 7.1 : Speed-up of GPP,GPU[126],FPGA for the Smith-Waterman algorithm with affine gap penalty (GPD=-10, GPE =-2). The homology search is performed using a query sequence of length 256 residues and database of 513,877 sequences of 188,625,310 residues

Platform	Execution time(s)	CUPS (Giga)	Speed-up
FPGA	1.8	26.8	269:1
GPU	21.2	2.3	23:1
GPP	483.9	0.1	1:1

Profile-HMM based sequence alignment is a position-specific sequence alignment which is useful in searching for homology among distantly related sequences. It uses the well-known DP-based Viterbi algorithm to search for the optimal path of the sequence-to-profile HMM search. The corresponding 'software only' implementation of the Viterbi algorithm is available in the HMMER package with HMMER 3.0 as its latest software package. In this analysis, the HMMER was executed on the 2.0 GHz desktop computer with an Intel Dual Core Processor (E6600), which is the same as the one used to execute the SSEARCH software for the Smith-Waterman algorithm. In the case of the GPU platform, the work reported in L. Xiaoqiang et al. in [128] is used as reference. During the experimental work, the FPGA was clocked at 150 MHz. All computation platforms were based on the same length (a motif length of 215) of the profile HMM and the input sequence database was taken from the NCBI non-redundant database which comprises of 15.2 million subject sequences [130]. The implementation results of the respective platforms are presented in Table 7.2. It is noted that the FPGA performance in terms of CUPS and speed-up remains the highest, followed by the GPU implementation.

Table 7.2: Speed-up of FPGA and GPU [128] against GPP for HMMER profile HMM length 215 against database of 15.2 million sequences or 4,560,000,000 [129].

Platform	Execution time(s)	CUPS (Giga)	Speed-up
FPGA	33.2	29.5	8.3:1
GPU	62.5	15.7	4.4:1
GPP	275.0	3.6	1:1

Finally, the heuristic-based sequence alignment is used to evaluate FPGA performance against other platforms, particularly in the case of the gapped BLAST with the two-hit method. The XC5VLX110-3FF1153 device was used in this test with all of the three BLAST stages (seed generator, ungapped extender and gapped extender) were implemented on the Xilinx Virtex-5. The designed core was clocked at 200 MHz with the aforementioned BLAST stages ran in parallel in order to gain higher computational performance. In the case of the GPU implementation, the work presented in [131] is used as reference. The BLASTp algorithm was implemented on the NIVIDIA GeForce GTX

295 GPU with a hybrid parallelization scheme. On the other hand, the corresponding GPP solution was based on the NCBI BLASTP software execution as reported in [131]. In the reported work, the NCBI BLASTP version 2.2.22 was executed using four threads on the Intel Quad-Core i7-920 CPU, 2.66 GHz. For each of the implementation platforms, a query sequence accession no. P42018 of length 254 and a database sequence of 9,230,955 sequences or 3,163,461,953 residues were used to evaluate performance of each implementation platform. The implementation results of the respective platforms are summarized in Table 7.3. The CUPS figures show that the FPGA remains the superior platform.

Table 7.3: Speed-up of GPP [131], GPU [131], FPGA for gapped BLAST with two-hit method of 254 residues length against 9,230,955 sequences or 3,163,461,953 residues

Platform	Execution time (s)	CUPS (Giga)	Speed-up
FPGA	2.98	24.2	12 :1
GPU	5.90	12.9	6:1
GPP	36.0	2.9	1:1

7.1.3 Power and energy consumption

In this section, the analysis of the power utilized by each implementation platform is based on the amount of power used during the sequence homology search of the accelerated implementation platforms (FPGA and GPU). This is because the processing task of these platforms is performed on the respective accelerator card rather than in the host processor, whereby the host is only used for data transfer operations. Details of power consumption of each platform are shown in Table 7.4.

Table 7.4: Power and energy consumption of the Smith-Waterman implementation on FPGA, GPU [126] and GPP.

Platform	Power (Watt)	Energy (Joule)
FPGA	28	50
GPU	96	2035
GPP	130	62907

In the case of GPU implementation in [126, 132], no power consumption was reported, therefore, the typical power consumption of the GeForce 9600GT was taken from the manufacturer's technical specifications, which is available online[127]. In this evaluation, the Smith-Waterman algorithm is first presented followed by the other two algorithms towards the end of this chapter to further evaluate the efficiency of FPGA as compared to other computing platforms. Energy consumption was calculated by multiplying the power consumption of each platform with its execution time obtained from Table 7.1. Based on the results calculated as presented in the third column of Table 7.4, the FPGA achieves energy efficiency of three orders of magnitude higher than the corresponding GPP implementation. The GPU comes second with one order of magnitude, energy efficiency higher compared to the GPP. In addition, performance per watt values for each platform can be calculated by dividing the CUPS figure in Table 7.1 with the power consumption presented in Table 7.4. The calculated CUPS performances per watt are presented in Table 7.5. The results show that the FPGA solution remains the more energy efficient platform, followed by the GPU.

Table 7.5: Performance per watt figures of the Smith-Waterman algorithm on FPGA, GPU [126] and GPP.

Platform	Performance (MCUPS) per watt	Normalized performance per watt
FPGA	958.1	1197.6
GPU	23.7	30.0
GPP	0.8	1

7.1.4 Cost

Cost is another important criterion which affects decisions on the choice of implementation platforms for accelerating user applications. In an effort to determine performance per unit cost in the dynamic programming-based biological sequence alignment, the work reported in [133] was used as reference in estimating the purchase and development costs associated to the FPGA, GPU and GPP platforms. The purchase cost includes the cost of purchasing the host, while the development cost was based on the salary of a newly-graduated student of 20 USD per hour. The costs associated with each platform are shown in Table 7.6. In terms of development time, an FPGA

programmer requires more time to design the Smith-Waterman algorithm in hardware than others. This includes learning specific hardware API and FPGA debugging, which accounts for about 80 percent of the total time [133]. This contributes to 50x higher FPGA costs as compared to the GPP solution. GPU comes second with 8x higher cost than the GPP.

Table 7.6: Development time in days and total cost (purchase and development cost) of the Smith-Waterman algorithm implementation on the FPGA, GPP and GPU [133]

Platform	Development time (Days)	Purchase Cost ($) Including Host	Development Cost ($)	Total Cost ($)	Normalized total cost
FPGA	300	10,000	48,000	58,000	50
GPU	45	1450	7,200	8,650	8
GPP	1	1000	160	1,160	1

To evaluate performance per unit dollar spent, the CUPS figures in Table 7.1 are divided by the total cost associated with the respective implementation platforms in Table 7.6. The results show the FPGA to be a more economic solution on the basis of performance per dollar spent as compared to the other technologies. Since each platform was manufactured with the same process technology (65nm), thus the normalized figure has evaluated each core performance fairly.

Table 7.7: Performance per dollar and per watt for the FPGA, GPU implementation of ref. [126] and the GPP implementation on a comparable desktop computer

Platform	Performance (MCUPS) per dollar	Normalized performance per dollar spent
FPGA	0.46	5.1
GPU	0.26	2.9
GPP	0.09	1

In the case of the HMMER and the BLAST architectures, the same normalization procedure is used to evaluate both the energy efficiency and cost effectiveness of the FPGA solution. Based on the CUPS figures as in Table 7.2 and Table 7.3, the normalized CUPS performance per watt is calculated. In terms of energy efficiency,

FPGA remains the highest with normalized CUPS performance of 39x and 56x higher than the GPP solutions for the HMMER and the BLAST algorithms respectively. The GPU platform comes second with normalized CUPS performance of 6x (HMMER) and 8x (BLAST) performance higher than the software implementation. In the case of cost effectiveness, the development cost of the respective platform as in Table 7.6 is taken into consideration for normalization. From the calculated results, it is noted that GPP solution comes first with an order magnitude higher than GPU and FPGA for both HMMER and BLAST algorithms. The higher development costs of the respective core architectures and the hardware resources limitation are the main factors which cause the performance degradation. Compared to the GPP solution, the development time of the FPGA is 300x longer. This due to the optimized core architecture has been carefully designed using the low level language to benefits from parallelism in the FPGA. This has led to the cost inefficiency of the FPGA solution when normalized against the significantly higher development time compared to others. However, among its other advantages, the HDL-based core architectures can be re-used for other types of sequence alignment or other systolic array-based architectures. This will shorten the development times, since the cores have been developed and optimized in this work for code re-use. Moreover, the pure HDL core architectures enable designers to have full controls in managing hardware resources to achieve higher degree of parallelism. In the context of hardware limitation, the designed architectures can be redeployed onto higher logic density FPGAs with minimal design effort in order to achieve higher degree of parallelism.

Alternatively, with the emergence of the AutoESL in 2012, FPGA development time can be significantly reduced by designing user applications using higher level languages. AutoESL stands for auto electronic system level (ESL) and it is a high level synthesis tool company founded in 2006 [134]. After three years of research and development, AutoESL unveiled a C-to-gate synthesis tool known as AutoPilot. AutoPilot was developed in the University of California, Los Angeles (UCLA) and this tool was initially proposed at the Design Automation Conference (DAC 2009) [134]. The ability of the high level synthesis tool to convert C, C++ and SystemC languages to equivalent logic gates enables FPGA designers to capture their designs using high level language rather than HDL [134] with design time as fast as the GPP solution. In April 2012, Xilinx Inc. has announced the autopilot/Vivado High Level Synthesis (HLS) tool

for commercial and academic use [139]. By using this C-based FPGA design entry, the FPGA development time significantly reduced and for the case of the Virtex-5 XC5VLX110 FPGA, it is expected to gain up to ~8x and ~12x normalized CUPS performance per unit dollar spent higher than the GPP solution in the case of HMMER and BLAST acceleration respectively. In this sense, the proposed HDL-based architectures can be used as reference designs in evaluating efficiency of the HLL-based sequence alignment core architectures. Higher computational performance is achievable, when the core is implemented on the Zynx-7000 family. This Xilinx all programmable SoC platform combines the ARM dual-core Cortex-A9 processor with the 28nm programmable logic onto the same silicon die for ease of co-processing applications in embedded computing platform [135]. In the case of targeting this platform for implementing sequence alignment core architecture, the PE systolic arrays can be implemented onto the programmable logic (PL) region; while the control intensive operation and database transfer tasks can be executed by the processing system (PS) i.e. the ARM processor. The AXI bus, which is a part of the advanced microcontroller bus architecture (AMBA) provides high bandwidth data transfer between the PL and PS regions to support the co-processing operation. The Zedboard is example of the Xilinx all programmable SoC platform. With cost of only USD 319.00 [136], this low cost development board can be used to implement the aforementioned sequence alignment algorithms with expected higher normalized CUPS performance per dollar spent.

7.2 Summary and conclusions

In this chapter, evaluations of FPGA performance against the widely-used sequence alignment implementation platforms i.e. GPU and GPP were conducted. This comparative study covered FPGA efficiency in terms of speed, area, power, energy and costs associated with the aforementioned platforms. First, the cell update per second (CUPS) and speed-up figures of the Smith-Waterman with the affine gap penalty, the profile HMM-based sequence alignment and the heuristic-based sequence alignments were calculated. The FPGA solutions for the respective types of sequence alignment were based on the proposed reconfigurable architectures, while the GPU implementations were based on reported implementations in the literature. For the Smith-Waterman with the affine gap penalty algorithm, the FPGA solution achieved a

two orders of magnitude higher speed-up than the GPP, followed by the GPU. The latter yielded an order of magnitude speed-up performance higher than the SSEARCH version 35.05. The 'software only' implementation was executed on the 65nm Intel Dual Core processor (E6600) with 2.0 GHz clock frequency. In the case of the profile HMM-based sequence alignment, both FPGA and GPU achieved an order of magnitude speed-up higher than the GPP counterpart, with the FPGA speed-up of 8x followed closely by the GPU a with speed-up of 4x against the latest HMMER package version 3.0. Finally, for the heuristic-based sequence alignment, the FPGA implementation of the gapped BLAST with the two-hit method achieved a speed-up of 12x, while GPU yielded a 6x speed-up as compared to the NCBI BLASTP version 2.2.22 which ran on the Intel Quad-Core i7-920 CPU.

For the evaluation of power and energy consumption, the FPGA-based Smith-Waterman algorithm with the affine gap penalty achieved lower energy consumption by three orders magnitude as compared to the GPP. The GPU implementation came second with an order of magnitude lower than the GPP. This showed that the FPGA remains the most energy efficient platform. However, in terms of the operational costs of each implementation platform, the FPGA required substantially higher costs of 50x, followed by the GPU with 8x higher cost compared to the GPP implementation. The costs associated with each implementation were based on both the purchase cost of the host and development costs. The longer development time of FPGAs was due to the nature of the lower abstraction level of HDL and requiring experience and highly skilled hardware programmers as compared to GPUs and GPP platforms. In the context of performance per dollar spent, FPGAs remains the superior platform compared to the others, with normalized performance per dollar spent of 0.46 million CUPS as compared to GPP. The GPU came second with 0.26 million CUPS per dollar spent.

This evaluation demonstrated that FPGAs are an energy efficient and cost effective platform for implementing biological sequence alignment. This is due to the higher degree of parallelization offered by FPGAs. Ultimately, it can be concluded that FPGAs are eminently viable alternative platform for biological sequence alignment. Future research in the area of sequence alignments and bioinformatics should be directed towards optimizing FPGA solutions and implementing them on higher logic density FPGAs for even higher degrees of parallelism.

Chapter 8
Summary, Conclusions and Future Work

8.1 Summary and Conclusions

This thesis describes the use of state-of-the-art reprogrammable system-on-chip technology in the form of an FPGA platform to accelerate the three widely-used sequence alignments;

- The Smith-Waterman with affine gap penalty (optimal)
- The Profile hidden Markov model (heuristic)
- The Gapped BLAST with the two-hit method (heuristic)

Sequence alignment is a time-consuming and repetitive task. To perform homology searches against sequences in a database, each search requires a matrix filling operation which is the most time-consuming part of sequence alignment. Any sequence with the highest score gives invaluable clues as to the function of genes, and this can eventually which eventually lead to breakthroughs in drug engineering, the construction of phylogenetic trees, and many other scientific advances. However, the matrix filling operations require quadratic time complexity if executed using a general purpose computer. Over the last decade, a linear systolic array has been implemented in hardware to accelerate dynamic programming-based sequence alignment. However, due to the limited logic and memory resources in hardware, maximum parallelization is prohibitive. This is because of a proportionally higher number of PEs are required for such alignment matrix operation, since only one PE can be used to hold a query sequence residue at a time.

Alternatively, PE systolic arrays with folded architecture have been used, as reported in literature. Processing query sequences longer than physically implementable PE systolic arrays in hardware is achieved by reusing the PE systolic arrays for subsequent pass computations. Although such multiple-pass computation promotes silicon reusability and optimizes hardware resources, PE configuration time increases

proportionally with the number of passes. This is because each PE requires a new set of coefficients in between multiple-pass processing. Run time reconfiguration (RTR) is another method which reconfigures PEs on the fly. It generates any number of PEs to suit the length of a query sequence. Among other configuration ports available, the ICAP port is commonly used to configure partial bitstreams (blocks of PEs with their corresponding coefficients) into the FPGA as it offers the highest data transfer rate of 3.2 Gbps. However, this maximum data transfer rate is not capable of coping with the speed requirements of sequence alignment operations. Due to the limited bandwidth of the ICAP, this alternative is less attractive and no further RTR-based sequence alignments have been reported. Therefore, in this research, PE systolic arrays with an efficient strategy to schedule between alignment matrix computation and CE configuration have been proposed in the core architectures of the respective sequence alignments. This overlapping strategy is introduced into sequence alignment in order to overcome both the time and area complexity of sequence alignment and in this research the strategy is referred to as overlapped computation and configuration (OCC). The use of linear systolic arrays successfully reduced the computation time to linear time complexity as compared to 'typical software' only simulation. Additionally, with the OCC-based sequence alignment core architectures, both the time and area complexity of the respective types of sequence alignment were significantly improved. The bounded PE configuration time enables this overlapping operation to continue so that subsequent pass computation occurs smoothly without interruption, hence increasing overall system throughput. This overlapping operation virtually removed the time required for PE configuration with the added advantage of having a fixed number of only two CEs per PE.

In this research work, the respective core architectures were prototyped on the Alpha Data ADM-XRC-5LX card with Virtex-5 XC5VLX110 FPGA on it. The implementation results showed that the proposed architectures achieved CUPS performances of 26.8, 29.5, and 24.2 GCUPS respectively for the dynamic-programming based local alignment, profile HMM-based sequence alignment and the gapped BLAST with the two-hit method. In terms of speed-up improvements, a comparison was made from two different perspectives; comparison against software implementation and reported FPGA implementations. In the case of the software implementation, the DP-based implementation achieved an average speed-up of more than 200x as compared to

the SSEARCH 35 software. In the context of the profile HMM-based sequence alignment, the designed core achieved 103x speed-up in the case of HMMER 2.0 software and an average speed-up of 8x for the latest version, HMMER 3.0. On the other hand, the gapped BLAST with the two-hit method achieved more than a tenfold average speed-up compared to the latest NCBI BLAST software version 2.2.27+.

The reported FPGA implementations on the respective sequence alignments used different types of FPGA to implement their designs. Therefore, to evaluate the proposed architectures fairly against the others, the raw speed-up performance was normalized according to area and lithography technology used. This new performance metric was derived from the primitive element in the FPGA i.e. the logic cell (LC). LCs were used rather than CLB slices because the former effectively normalized in terms of area utilization in the FPGA regardless of the types of FPGA used. In addition to the primitive element, the ratio of internal look-up table delay for the respective FPGAs was also taken into consideration. The normalized performance indicator removed the inherent advantages of the area and lithography technology of a particular FPGA, resulting in fairer speed-up comparisons. Based on the proposed metric, the first architecture achieved more than 50 percent improvement, while the HMMER acceleration in hardware gained a normalized speed-up of 1.34 compared to other FPGA implementations. Finally, in the case of the BLAST sequence alignment, the designed core achieved 11x speed-up against other FPGA implementations after taking into account the advantages of the Virtex-5 FPGA. It is worth mentioning here that the speed-up performance comparisons for each type of sequence alignment were based on various passes computations due to the limited hardware resources in the FPGA used. Higher computational performance can be achieved by redeploying the proposed cores on newer, bigger and faster FPGAs. This can be done with minimal design effort as all designs were captured purely in Verilog HDL and the PE systolic arrays are scalable to any number depending on the FPGA chip in hand.

The final evaluation considered the suitability of FPGAs to become a viable alternative in bioinformatics, particularly in biological sequence alignment. Therefore, evaluations of FPGA against the GPP and GPU as acceleration platforms for biological sequence alignment were carried out towards the end of chapter 7. The evaluations covered FPGA efficiency in various aspects including development time, power consumption and operational costs. The performance of a designed core is greatly

influenced by the skill and experience of the HDL programmer as well as the type of FPGA used to implement the design. In terms of development time, all of the designed architectures required more time as compared to other platforms. However, the higher parallelism of the linear systolic array and the OCC architectures showed that FPGA-based sequence alignment cores were the most energy efficient platforms i.e. 958.1 MCUPS per watt. The GPU implementation came second with 23.7 MCUPS per watt, followed by the GPP with 0.8 MCUPS per watt. In the context of performance per dollar spent, FPGA-based sequence alignment cores remained the best platform compared to the others. Their normalized performance per dollar spent was 5.1 million CUPS followed by the GPU at 2.9 million CUPS per dollar spent compared to the GPP solution. These results show that FPGAs can be a viable alternative which offer a smaller area footprint, an economic 'green' solution and cost effectiveness compared to the other acceleration platforms.

8.2 Future Work

The proposed reconfigurable architectures have been captured using Verilog HDL to enable HDL-based customizations so as to suit different requirements of sequence alignment algorithms. In this section, further suggestions are presented beginning from the initial step to more sophisticated approaches for future improvements in implementing sequence alignment algorithms in hardware. Sequence alignment is a fundamental tool in molecular biology and further improvements towards the ease of usability and programmability to the mainstream programmers is hopefully will bridge the gap between hardware designers and molecular biologists.

8.2.1 Prototyping on denser FPGAs

The initial step towards higher performance sequence alignment accelerators is to increase the degree of parallelism. Current core architectures are scalable to any number of PEs and are not restricted to a particular FPGA, and therefore the implementation of reconfigurable architectures onto higher density FPGAs can be done with minimal design effort. In the case of the hardware implementation of the Smith-Waterman algorithm, the XC5VLX110 device occupied up to 140 PEs. Higher density FPGAs such as the XC5VLX330 or XC6VLX760 FPGA, offering 3x or 7x higher logic density can

be used to increase system performance without compromising the sensitivity of the homology search.

In the case of profile-to-sequence alignment, the reconfigurable architecture requires more logic resources in order to implement the Viterbi algorithm in the PE as compared to the first architecture. Hardware implementation on the XC5VLX110 device showed that the maximum achievable number of PEs was 40. For higher computational performance, the core architecture can be implemented onto the latest Xilinx FPGAs such as the XC72000T FPGA which offers up to two millions logic cells, 118,560 slices and 720 blocks RAM. The resulting PE systolic arrays after taking into account all of the other modules in the core's architecture, are expected to offer up to 1,000 PEs.

For the gapped BLAST with the two-hit method architecture, the core has three different blocks of PE systolic arrays for the seed generation, ungapped extension and gapped extension stages. Current hardware implementation requires intensive communication between these stages and the number of PE systolic arrays greatly affected overall performance of the core. Since the design allows for the scalability of PE systolic arrays at each stage, prototyping the proposed core into higher logic density FPGAs will therefore increase the degree of parallelism at each stage and consequently increase the overall system performance.

A more sophisticated approach to the acceleration of sequence alignment algorithms can be implemented by integrating FPGA with other computing platforms such as the GPU and GPP in the so-called heterogeneous computing. This way, overall system architecture can benefit from exceptional advantages of each type of platform. The following section first discusses the integration of an FPGA with a processor platform in the SoC-based Xilinx FPGA, followed by an explanation of the advent of heterogeneous or hybrid computing. Then an adaptive computing strategy is elaborated before the closing remarks of the thesis are presented.

8.2.2 System on Chip-based computing

The System on Chip, Zynx™-7000 family can be used to provide the co-processing infrastructure for bioinformatics applications. The industry's first extensible processing platform (EPP) tightly integrates both the low power reconfigurable logic fabric, i.e. the FPGA and the 28-nm ARM(R) Cortex™-A9 microprocessor-based system onto single

silicon die in order to enable low power and cost effective co-processing solutions for high-end embedded applications [137]. In the case of biological sequence alignment, this programmable System on Chip (SoC) can be used for controlled-intensive sequence alignment algorithms such as the gapped BLAST with the two-hit method. Currently, the complex controllers of the BLAST stages were implemented in FPGA and the main limitation of the proposed architecture is the intensive and complex control mechanism involved in the BLAST stages. The migration of this controller into the dedicated ARM processor can overcome this problem and the FPGA can be dedicated to perform the massive parallel processing of the PE systolic arrays for each of the BLAST stages. With the high-bandwidth AMBA^{R} 4 Advance Extensible Interface (AXI4^{TM}) available in this latest Xilinx Zynx™-7000 SoC, the common performance bottlenecks of typical hardware software co-design such as in control and data transfer operations are eliminated. Therefore, the control and data transfer operations between the FPGA and the processor work seamlessly together to enable higher computational performance with energy and cost effective solutions.

8.2.3 Hybrid computing

GPUs and FPGAs are among the most promising platforms for high performance computing applications. This is due to the cost effectiveness and energy efficiency they offer compared to off-the-shelf-microprocessors. The latter require a specialized power supplies or cooling facility to achieve teraflop/s performance as high as FPGAs and GPUs. This is due to the power consumption issue; the so-called power wall of the CPU-based high performance computational platforms such as supercomputers and other types of computer clusters. Consequently, heat dissipation increases proportionally with clock speed, thus pushing the microprocessor into the so-called speed wall. For example, Intel cancelled their 4GHz processors due to these issues and came up with alternative multi-core architectures [138], where the processor cores are integrated onto the same silicon die. To date, the development time associated with FPGA and GPU acceleration platforms has been considered relatively high compared to that of the CPU, and this is especially true for the FPGA. Interestingly, in April 2012, Xilinx unveiled the Vivado Design Suite, which enables mainstream programmers to use C, C++ or SystemC to capture their design into FPGA [139]. This bridges the gap between software

programmers and hardware designers, and thus, the exceptional advantages of FPGAs, GPUs and GPPs can be shared in a single-language programming heterogeneous computing platform in order to achieve higher computational performance, especially for scientific computing applications.

In the case biological sequence alignment, the core architectures can be designed using HLL for ease of the implementation of the aforementioned hybrid computing platform. An initial step might be to compare the performance efficiency of the HLL-based core architectures with the HDL one. This will hopefully enable the use of HLL to program the FPGA, with results at least equally efficient to those implemented using the HDL-based design. The use of HLL to implement FPGA designs helps accelerate the time to market. Another step would be to implement the HLL-based core architecture onto heterogeneous computing platforms in order to benefit from the advantages of FPGA, GPU and GPP towards realizing a co-processing infrastructure. This should give higher computational performance and a better parallel implementation of biological sequence alignment.

8.2.4 Adaptive computing

Adaptive computing offers efficiency in hardware resource utilization based on computational demands and resources availability. In the case of the proposed architectures, this computing strategy would allow the use of PE systolic arrays to be optimized, whereby PEs will be generated based on computing demand rather than unused PEs remaining in idle mode. For instance, the gapped BLAST with the two-hit method has three different stages, each of which requires different numbers of PEs. The current design has fixed PE systolic arrays for the different stage during its operational mode. For most of these stages, especially the ungapped and gapped extender stages, only smaller PE systolic arrays are required for alignment matrix computation, while the remaining PEs remain idle. Therefore an adaptive resource allocation based on adaptive computing strategy can be developed to dynamically allocate parallel execution based on computing demands as well as the availability of processing elements rather than leaving them unused. This run-time parallelism strategy is sometimes referred to as invasive computing and details of such a strategy and its applications have been extensively reported in [140], [141]and [142].

8.3 Closing remarks

Sequence alignment is the first step towards the in-depth exploration of many other complexes and area of study in bioinformatics. This includes the identification and quantification of conserved regions, the profiling of genetic diseases, phylogenetic analysis and micro-array experimentation. The advent of single-language programming heterogeneous computing platforms is gradually bridging the knowledge gap between hardware designers and biologists. Thus, tremendous growths of research in this multidisciplinary area are expected in the future as FPGA is now not only programmable by hardware designers, but also the mainstream programmers. The next step is to evaluate efficiency of the HLL-based designs to control and manipulate hardware resources to benefit from the parallelism offered by FPGAs. Ultimately, combinations between the hybrid computing platforms and the adaptive computing strategy will provide a promising future for bioinformatics applications such as in cancer research and many other complex problems in bioinformatics towards improving quality of life.

Appendix

Publications

A) Conferences

1. **M. N. Isa**, K. Benkrid, T. Clayton, C. Ling, and A. T. Erdogan "An FPGA-based parameterised and scalable optimal solutions for pairwise biological sequence analysis," presented at 2011 NASA/ESA Conference on Adaptive Hardware and Systems (AHS), 6-9 June 2011, pp. 344-351.

2. **M. N. Isa**, K. Benkrid and T. Clayton “A Novel Efficient FPGA Architecture for HMMER Acceleration" presented at 2012 International Conference on ReConFigurable Computing and FPGAs (ReConFig), 5-7 December 2012, pp. 1-6.

3. **M. N. Isa**, K. Benkrid and T. Clayton “A Highly Efficient Substitution Matrix Loader for Pairwise Sequence Alignment” presented at The 24th International Conference on Microelectronics (ICM), 17-20 December 2012, pp. 1-4.

4. **M. N. Isa**, K. Benkrid and T. Clayton “High Performance Gapped BLAST with the Two-hit Method Implementation on FPGA”, The Fourth International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies, HEART 2013, 13-14 June 2013 (Submitted)

5. C. Hong, K. Benkrid, **M. N. Isa** and X.Iturbe “Run-time Reconfigurable System for Adaptive High Performance Computing” The Fourth International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies, HEART 2013, 13-14 June 2013 (Submitted)

B) Journals/Letters

6. **M. N. Isa**, K. Benkrid, and T. Clayton, "Efficient architecture and scheduling technique for pairwise sequence alignment," *ACM SIGARCH Comput. Archit. News*, vol. 40, pp. 26-31, 2012.

An FPGA-based Parameterised and Scalable Optimal Solutions for Pairwise Biological Sequence Analysis

[1] M.N.Isa, [1]K.Benkrid, [1]T.Clayton, [1]C.Ling, [1]A.T.Erdogan.
[1]*System Level Integration Group (SLIg), Institute for Micro and Nano Systems (IMNS)*
School of Engineering, King's Buildings. The University of Edinburgh, Edinburgh, UK
(m.n.isa, k.benkrid, t.clayton, c.ling, ahmet.erdogan)@ed.ac.uk

Abstract

This paper presents the design and implementation of the computationally intensive Smith Waterman and Needleman-Wunsch algorithms on the Virtex-5 (XC5VLX110) FPGA (Field Programmable Gate Array). A parameterisable architecture of the processing element (PE) for the pairwise biological sequence analysis is presented. We evaluate our core's performance in terms of speed up performance and normalised cell update per second (CUPS) against various computing platforms including our previous Graphics Processor Unit (GPU) and other FPGA implementations. Our efficient hardware design produces the highest normalised throughput ever reported in literature.

1. Introduction

Biological Sequence analysis is one of bioinformatics and computational biology (BCB) applications recently gaining popularity in line with the rapid advancement in computing technology. Inferring relationship between sequences is vital in various applications including drug engineering, diseases diagnosis and construction of *phylogenetic* trees. Typical *'software only simulation'* running on a standard desktop computer produces result in hours or even days to finish especially when dealing with longer sequences [1]. In addition, with the increasing amount of sequence entries in the genomic database by a factor of 1.5 to 2 every year [2], scanning over sequence databases using standard computers is unable to produce results in realistic times. Therefore, hardware acceleration is crucial to provide results in a speedy and convenient way.

Heuristics and optimal approaches are the most widely used methods to perform sequence analysis. BLAST (Basic Local Alignment Search Tool) [3], Gapped BLAST and FASTA (Fast Alignment) [4] are examples of heuristic methods. They are relatively faster, due to their scanning method which are based on a sub-optimal approach that only search against related portions of genomic databases [4]. Another widely used alternative is the optimal technique. The Smith Waterman (local alignment) and the Needleman-Wunsch (global alignment) are the most popular algorithms used to align sequences for optimal results. In terms of accuracy, the optimal approach is the most sensitive approach as it uses an exhaustive search approach based on dynamic programming [5]. However, it is also more time consuming compared to heuristics.

Research presented in [6] reported that computing the Smith Waterman algorithm using the *software-only implementation* (SSEARCH34) for instance, consumes 98.61 percents of its time calculating the alignment score. It is the most time consuming operation involved in this algorithm and hence the need for acceleration. Parallel processing is a suitable technique to accelerate this operation in order to get results in realistic times [7]. In general, two potential approaches can be used to perform parallel processing; one is introducing systolic arrays in the sequence search algorithm, another is distributing the optimal alignment algorithms over networks of computers or supercomputers (scalable). The latter is referred to as coarse grain parallelism [8]. Previous research [9] has reported that systolic array approach is the best way to exploit fine grain parallelism. Systolic arrays essentially comprise of an array of tiny processors (often referred to as *Processing Element* or *PE*) arranged in an array form to compute scores in each residual pairs under comparison.

Rapid advancement in programmable logic and reconfigurable hardware has made acceleration of scientific computing applications possible. FPGA for instance, is among the promising computing platforms for scientific computing. Moreover, it offers faster development times, and lower Non-recurring Engineering (NRE) costs. In addition, with its re-programmable feature, developments of various biological applications are possible on the same silicon chip. However, FPGAs suffer from a relatively low level programming model as compared with off-the-shelf microprocessors (standard microprocessors or application specific microprocessors such as GPUs). This raises the need for optimised FPGA core implementations which is crucial for this technology to become

viable in scientific computing applications particularly in biological sequence analysis.

The remainder of this paper will discuss background on sequence homology search, followed by a discussion of a number of related works. Our design and implementation strategy are then presented. Comparison and evaluation of our implementation are also discussed before conclusion and future plans are laid out.

2. Background

Biological sequences diverge from a common ancestor by the process of *mutation* and *selection* [10]. The process of *mutation* for instance, includes residues' change in a sequence (*substitution*), addition (*insertion*) and removal (*deletion*) of residues. *Insertions* and *deletions* are referred to as *gaps* when aligning sequences. Sequence Alignment is a well-known technique used to analyse the changes of the above-mentioned biological processes. It is essentially a process of comparing biological sequences (e.g DNA, RNA or protein) with an objective to find *homology* between them. Figure 1 illustrates two examples of DNA sequences with x and y being the query sequence and the subject sequence with lengths 12 and 14, respectively.

x: c g g g g t a t c c a a

y: c c c t a g g t c t c c c a

Figure 1. DNA sequence x and y

Aligning these two sequences may result in different ways of alignment as shown in Figure 2 with gaps represented by '−'. A score is typically used to measure the degree of similarity for each of the possible alignments shown in Figure 2. Any sequence in the database with the highest score will be chosen as the best match. From a biological point of view, the selected sequence is likely to share common functional, structural, or evolutionary relationships with the query sequence [10]. The score for each residue pair is usually represented in an *alignment matrix* which maps a relationship between the query sequence and subject sequence in the form of residual pair's scores. Each of the score essentially represents the highest score among the *substitution*, *insertion* and *deletion* scores modelled from the biological processes mentioned earlier. The *substitution* scores are obtained from the amino acids probabilistic model. This model is a two dimensional matrix representing the relationship between amino acids (protein residues), known as *substitution matrix*.

x: c g g g t a − − t − c c a a

y: c c c − t a g g t c c c − a

x: c g g g t a − − − t c c a a

y: c c − − c t a g g t c c c a

x: c − g g g t a − − t c c a a

y: c c − − t a g g t c c c c a

Figure 2 : Alignments for sequence x and y

The *BLOSUM50*, *BLOSUM62* and *PAM* are examples of such matrices. A *gap* in sequences under comparison is undesirable and thus penalised with a certain value known as *gap cost*. There are two different gap models widely used [10]; linear and affine gap penalty models. A gap cost in the linear gap model is a constant value, but for the affine gap model, the gap cost is determined by a function. The latter comprises of *gap-open (d)* and *gap-extension (e)*. The gap-open is a gap cost used when opening a new gap in a sequence and the subsequent gaps are penalised linearly (*gap-extension*). A standard gap penalty cost associated with a gap of length g is given either by (1) for linear gap penalty or (2) for affine gap model.

Linear Gap Penalty

$$penalty(g) = -ge \tag{1}$$

Affine Gap Penalty

$$penalty(g) = -d - (g-1)e \tag{2}$$

3. Biological sequence analysis: Optimal alignments approach

Two types of well-known optimal alignment algorithms are the Smith-Waterman (S-W) and Needleman-Wunsch (N-W). Both alignment techniques use dynamic programming to compute the score of the alignment matrix ($F(i, j)$). The former algorithm focuses on the subsequence similarity between two sequences while the latter tries to align entire sequences.

3.1. The Needleman-Wunsch algorithm

The Needleman-Wunsch algorithm as shown in (3) was introduced by Needleman and Wunsch in 1970 [10]. It is used to find a global alignment score between two sequences. This algorithm searches for entire alignment of a query sequence x, and a subject sequence y.

$$F(i, j) = \max \begin{cases} F(i-1, j-1) + s(x_i, y_j) \\ F(i-1, j) - d \\ F(i, j-1) - d \end{cases} \tag{3}$$

3.2. The Smith-Waterman algorithm

The Smith-Waterman algorithm [10] was proposed in 1981 by T. F. Smith and M. S. Waterman. The recursion equation is shown in (4).

$$F(i,j)=\max\begin{cases}0\\ F(i-1,j-1)+s(x_i,y_j)\\ F(i-1,j)-d\\ F(i,j-1)-d\end{cases} \qquad (4)$$

It is an optimal algorithm used to compute local alignment scores. Unlike the global alignment, this alignment searches for the best alignment between sub-sequences of x and y. In contrast, both equations are identical, except zero is added to the maximum expression in the case of local alignment. This will make local alignment scores saturate to zero whereas global alignment scores can take negative values.

3.3. Dynamic programming

The dynamic programming technique was formalised by Richard Bellman [11] and well-known as a powerful technique to solve problems in time complexity of $O(n^2)$ and $O(n^3)$. Theoretical time complexity when comparing two biological sequences with lengths of m and n residues is $O(m*n)$. When both $|m|$ and $|n|$ are equal, then the running time becomes $O(n^2)$. By mapping the dynamic programming into an array of systolic processors, the computation complexity significantly reduced to $O(m+n-1)$. It is much like a "divide-and-conquer" technique except it allows overlapping of sub-problems. In the case of local and global alignments, it breaks down the recursive equations into a reasonable number of smaller sub-problems. These sub-problems are then solved with the optimal solutions and ultimately give optimal solutions to the main problem. Since both local and global alignment algorithms are almost identical, we start explaining dynamic programming by illustrating the alignment matrix for the global algorithm as shown in Figure 3. The $F(i,j)$ is indexed by i and j with one index per sequence character. The three adjacent elements are used to compute the score of $F(i,j)$. The ultimate score for the current cell ($F(i,j)$) is the highest score from any of three possible alternatives;

- The diagonal element $F(i-1,j-1)$,
- Top element $F(i,j-1)$,
- The left element $F(i-1,j)$.

These three elements are required to compute the alignment matrix $F(i,j)$ of i_{th} residue of sequence x and j_{th} residue of sequence y. The $s(x_i,y_j)$ is their corresponding substitution matrix score. Before constructing the alignment matrix, boundary values are required. The $F(0,0)$ is set to zero as it obviously does not represent any alignment either in sequence x or y. It is thus always set to zero for both cases. For the case of global alignment, $F(i,0)$ which represents alignment of prefix x to all gaps in y, must be set to $-id$. Similarly for the $F(0,j)$, which is set to $-jd$ as it represents alignment of prefix y to all gaps in x direction. In the case of local alignment, we are only interested to align a subsequence portion between two sequences and hence all boundary values ($F(i,0)$ and $F(0,j)$) are set to zero. Computation of the current score $F(i,j)$ starts from the top left of the similarity matrix as illustrated in Figure 3. This matrix is then built up recursively from the first segment $x_{1...i}$ of x up to x_i and the first segment $y_{1...j}$ up to y_j.

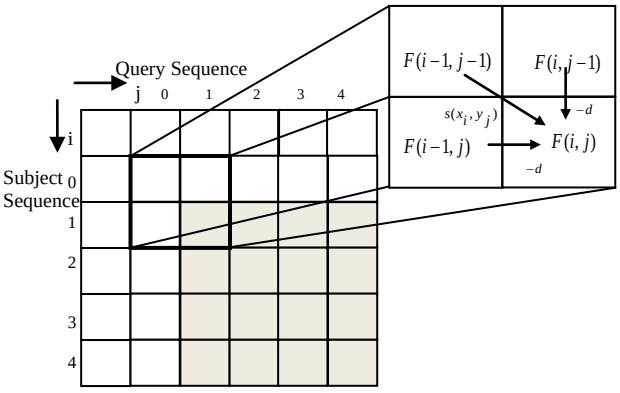


Figure 3. Computing $F(i,j)$ in the alignment matrix

3.4. Aligning sequences with more accurate results

In linear gap penalty, the gap penalty is equal to the number of gaps times a constant, whereas for the affine gap, the gap penalty for the overall sequence depends on the affine gap function as in (2). The latter is more realistic and therefore aligning sequences with this gap penalty produces more accurate result. The recursive equation for the global alignment with affine gap penalty is shown in (5)[10].

$$F(i,j)=\max\begin{cases}F(i-1,j-1)+s(x_i,y_j)\\ I_x(i-1,j-1)+s(x_i,y_j)\\ I_y(i-1,j-1)+s(x_i,y_j)\end{cases}$$
$$I_x(i,j)=\max\begin{cases}F(i-1,j)-d\\ I_x(i-1,j)-e\end{cases} \quad (5)$$
$$I_y(i,j)=\max\begin{cases}F(i,j-1)-d\\ I_y(i,j-1)-e\end{cases}$$

Another algorithm to implement the alignment if the lowest mismatch scores below $-2e$, which only involves two states is given by (6)[10].

$$F(i,j)=\max\begin{cases}F(i-1,j-1)+s(x_i,y_j)\\ I(i-1,j-1)+s(x_i,y_j)\end{cases}$$
$$I(i,j)=\max\begin{cases}F(i,j-1)-d\\ I(i,j-1)-e\\ F(i-1,j)-d\\ I(i-1,j)-e\end{cases} \quad (6)$$

For the case of local alignment with the affine gap penalty model, zero is added to the maximum expression in both (5) and (6).

4. Related work

Although the optimal alignment method is capable to produce the most sensitive and accurate alignment, it suffers the slowest computation time compared to sub-optimal solutions introduced by heuristics approaches. A number of works have been proposed to implement these optimal sequence alignment algorithms. The Smith Waterman algorithm for instance, was implemented using *Instruction Systolic Arrays* (ISAs) approach as presented in previous study [12]. ISAs is a fine grain *n by n* mesh connected parallel processor consisting of 1024 processors on a single silicon chip designed to combine speed and simplicity of systolic arrays with flexible programmability. Other approaches which is based on Single Instruction Multiple Data (SIMD) concept are MGAP [13] and Fuzion [12]. Although both architectures are programmable, they have less PE density as compared to special purpose Application Specific Integrated Circuits (ASICs) [8]. Unfortunately, design and production costs for special purpose SIMD architecture are quite high [8] and consequently, no further productions have been reported. Work by [14] had seen the implementation of the same algorithm in *Systolic Accelerator for Molecular Biology Application* (SAMBA). SAMBA is a hardware accelerator for biological sequence comparison implemented using ASICs technology. The prototype demonstrated that the sequence search of 300 amino acids against SWISS-PROT-34 database consisting 21,210,389 residues was successfully done in 30 seconds [14]. Although this special purpose hardware offered higher PE densities, it is limited to only one particular algorithm (e.g the Smith Waterman algorithm) due to its non-programmable nature. In fact, a different chip is required for another algorithm (e.g the Needleman-Wunsch algorithm). These limitations led to the rise of reconfigurable hardware (RH) as a platform for biological sequence analysis.

Recent trends of computing technology have seen rapid advancement in RH such as FPGAs. Implementation of FPGA related bioinformatics and computational biology applications are extensively reported in [12], [15], [16], [17], [18], [19], [20], [21], [22], [23] including sequence alignments ranging from heuristics and optimal implementations. However, none of them clearly discussed their normalised design performance and only [17] reported their core capability to implement different types of algorithms into a single chip. In our work, we have implemented a library of biological sequence analysis that is able to perform local and global alignments by changing parameters during run time and at the end of this paper we also proposed independent performance evaluators to measure performance of our FPGA core against others.

5. Our hardware implementation

In this work, we develop a library of biological sequence alignment cores (Needleman-Wunsch and Smith Waterman) with the following parameters which can be set at run time.

(i) Types of *alignment algorithm*: Two different architectures could be implemented either the Smith-Waterman for local alignment or the Needleman-Wunsch for global alignment.

(ii) Types of *gap penalty*: This could be either an affine or linear gap penalty.

(iii) The *gap cost* : In the case of affine gap, any values either the gap opening (d) or gap extension (e) can be set.

(iv) The *match score*: It is the match score from substitution matrix attributed from the matched residue pair. The substitution matrix could be PAM, BLOSUM or any other protein probabilistic matrices.

(v) The *query sequence* length: The number of PEs is dictated by this length. Larger PEs could be inferred depending on the target FPGA. This core supports longer query sequences with it multi-pass architecture.

5.1. The linear systolic array

The linear systolic array for the pairwise sequence alignment is presented in Figure 4. The top diagram represents $F(i, j)$ element in each cell of alignment matrix. The $F(i, j)$ is computed using the dynamic programming technique mentioned earlier. For simplicity, we discuss the same length for both query and subject sequences ($m = n = 4$). Each column of this alignment matrix is equivalent to its corresponding PE as illustrated in Figure 4.

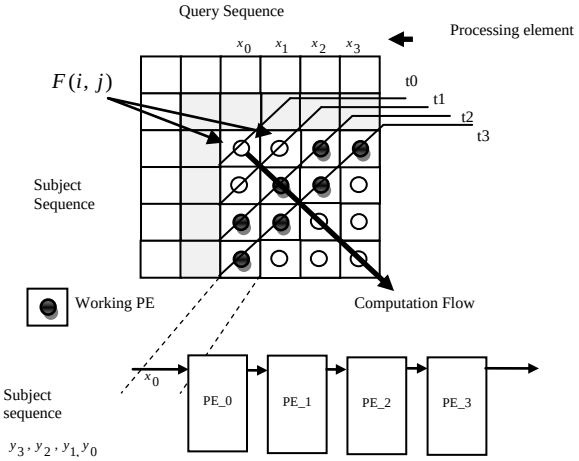


Figure 4. Alignment matrix with a linear systolic array

This implementation is based on a fine grain parallelism technique and consists of a pipeline of basic processing elements, each holding one query character or residue x_i whereas the subject sequence is shifted systolically through it. Each PE will have one query character and its corresponding substitution matrix's column in the form of a Look-Up Table (LUT). These two elements are preloaded to each PE during configuration state. During the running state, as the subject sequence is systolically shifted from one PE to another, the subject sequence character will become the selector from which to select the corresponding value of the $s(x_i, y_j)$. The diagonal arrow as illustrated in Figure 4 shows the computational flow starting from top to the bottom element of the alignment matrix diagonally. This significantly reduced the computation time complexity of ($O(n^2)$) to $O(m+n-1)$ where m and n are the lengths of the query sequence and subject sequence respectively. Details regarding the internal PE operations will be discussed in the next section.

5.2. Processing element architecture

Figure 5 details our generic Processing Element (PE) architecture for the affine gap penalty. It is able to perform either the local or global alignment depending on the *Cfg* (configuration) setting during runtime without regenerating a new architecture. This parameterisable feature also applies for the linear gap penalty architecture, which is reconfigurable on the same chip. The PE essentially comprises six instances;

(i) *Config. Unit*: A configuration unit is essentially a lookup table (LUT) which contains substitution matrix's column. This instance is important for loading both the match score ($s(x_i, y_j)$) and boundary values for the alignment matrix. *LUT Sel* is the selector for multiplexer which is connected to the subject sequence stream. It is useful to select the match score associated with the subject sequence from the LUT.

(ii) Adder: Used to add three diagonal elements with its corresponding substitution matrix score.

(iii) *MAX* Comparator: Performs three-way comparison of the diagonal elements.

(iv) D Flip-Flop: Temporarily stores the previously computed values (left and diagonal values) by means of delaying the data by one clock cycle.

(v) $I_X Gen$ and $IyGen$: Generate *insertion* values for I_X and I_Y respectively.

(vi) *Best Score*. Calculates the best score by means of comparing all input values to the current PE including score from the previous PE. This is important to find the highest score across all PEs. The best score determines the best matched subject sequence in the database.

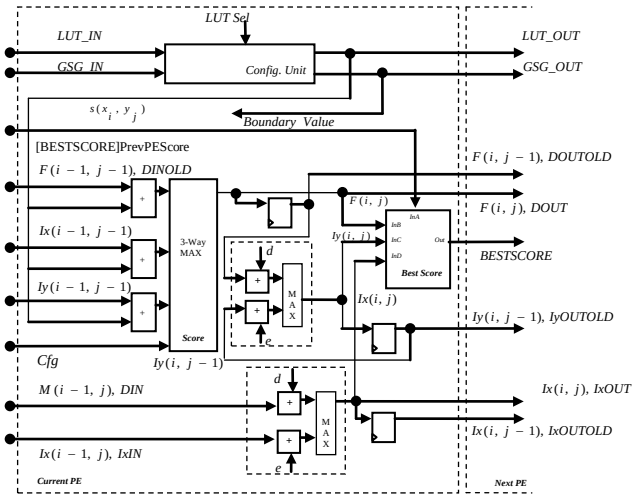


Figure 5. Our parameterisable PE architecture

Figure 6 illustrates a complete system of the biological sequence analysis processing core. Two main states involve in this core;

configuration and running states as mentioned in the previous section.

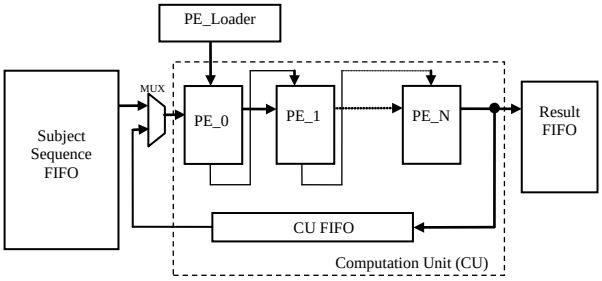


Figure 6. The biological sequence analysis core

During configuration time, the *PE_Loader* loads configuration data including a query sequence's character for each PE, boundary values and the corresponding substitution matrix's column with regards to the query sequence's character. All configuration data is loaded into the *Config. Unit* residing in each PE. Once the configuration is finished, the *Computation Unit* is triggered to execute the algorithm. At this stage, each of the subject sequence's characters residing inside the *Subject Sequence FIFO* starts flowing through each PE in every clock cycle. The score $F(i, j)$ in each PE is propagated from the first PE to the last one. The best score at the last PE of the current subject sequence under comparison is then stored in the *Best Score FIFO*. This process continues until the last subject sequence in the database. All scores for each subject sequence are stored inside the *Result FIFO*. Eventually, the best score among all scores will give the best match sequence from the database.

6. Result analysis

We evaluate our core's performance from two different point of views; (i) our core compared with our previous GPU implementation and (ii) our FPGA core compared to other FPGA implementations. Both evaluations were conducted against a variety of query sequence lengths ranging from 4 up to more than hundred characters or residues. The database sequence was extracted from the UniProt/SwissProt Knowledgebase, which comprises of 230,150 sequences with a total of 84,479,584 residues. Table 1 summarises all the set-up parameters involved.

Table 1. Parameters setting

Parameters	Settings
Alignment Algorithm	Local
Type of Gap Penalty	Affine
Gap Opening	-10
Gap Extension	-2
Substitution Matrix	BLOSUM 50
Word Length	11-bit

Our proposed core was implemented on the Alpha Data ADM-XRC-5LX card. This PCI (Peripheral Component Interconnect) Mezzannine Card (PMC) card has the Xilinx Virtex-5 FPGA on it. For the case of protein sequence processing either for the local or the global alignment, one PE utilised an average of ~88 slices on the Xilinx Virtex-5 FPGA to implement equation 5. Consequently, the XC5VLX110 device with 17,280 slices can easily fit up to 195 PEs. On the other hand, our previous GPU implementation uses an NVIDIA GeForce 8800 GTX GPU with 768MB device memory, 576MHz core clock frequency and 900MHz memory clock frequency. The code was developed using the NVIDIA SDK 9.5 and the CUDA 1.1 API. Figure 7 shows the results of our single-pass implementation (i.e all query sequence characters are fully fitted on a single FPGA chip) of the Smith Waterman algorithm implemented on our proposed core and GPU.

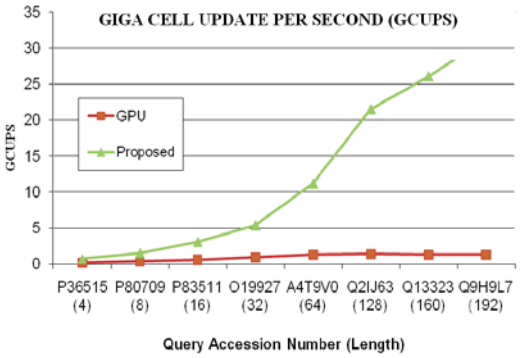


Figure 7. Comparison of our FPGA and our previous GPU in GCUPS

Both platforms use the same query sequence and database sequence as mentioned earlier, which is approximately 110 MB in size. The overall performance comparison of the protein sequence alignment is measured is Giga Cell Update per Second (GCUPS). The Cell Update per Second (CUPS) is a commonly used performance measurement in computational biology. A CUPS figure represents the number of alignment cells calculated per second of the affine gap penalty Smith Waterman including all additions, comparisons and the maxima computations [20]. Measuring the peak CUPS is done by multiplying the maximum frequency of the PE with the maximum number of PEs. The plot has demonstrated a proportional growth in processing speed as the size of the input query increases as compared to our previous GPU implementation.

Another method, which reflects the behaviour of a complete system (considers data transfer time, length of the query sequence and initialisation time) is also used to compare our purposed FPGA core with our GPU

implementation and the SSEARCH35. Table 2 presents our core speedup over the others. The speed up is calculated by dividing the execution time of our FPGA implementation over the GPU or the SSEARCH35. The results show an improved speedup with increasing sequence lengths.

Table 2. Speedup of our FPGA core over our GPU implementation and the SSEARCH35

Query Accession No.	Length	Speedup	
		Proposed vs. GPU	Proposed vs. SSEARCH
P36515	4	3.88	15.98
P80709	8	4.16	20.55
P83511	16	5.05	38.81
O19927	32	5.87	60.12
A4T9V0	64	8.49	149.38
Q2IJ63	128	15.48	276.89
Q13323	160	20.39	316.60
Q9H9L7	192	24.50	380.31

Performing fair and meaningful comparisons of our proposed core with other FPGA implementations is difficult as different technologies and performance measurements have been used by others. In an attempt to fairly evaluate our core with others, we use a new performance metric, which is effectively normalising with respect of different FPGA chips used (albeit test with area). This metric is called normalised CUPS/logic cell. CUPS is selected as it represents the sequence processing performance based on logic density and by getting the normalised CUPS, performance per elementary logic block could be calculated leading to independent of FPGA size. This new metric is calculated by dividing the CUPS performance with their respective number of logic cells utilised by the PEs. We use logic element (LE) or logic cell (LC) as the denominator to normalise the CUPS performance as it represents a fundamental metric for FPGA density. One logic cell comprises one 4-input look-up table and one register [24]. Based on the map report generated from the Xilinx ISE 13.1, the number of slices utilised (under the slice logic distribution summary) by our PEs is 17,228. Slice is made up from several logic cells and the number of logic cells per slice varies between different types of FPGA and vendors. For example, one slice in the Virtex-5 FPGA comprises of four LUTs, three dedicated user-controlled multiplexers, a dedicated arithmetic logic and four 1-bit registers. Altera uses ALM (Adaptive Logic Module) as their terminology to represent slice. One Altera ALM has seven LUTs, a dedicated arithmetic and a carry logic and two programmable registers [25]. These different internal architectures have led to our decision to used the logic cell rather than slice as the normalising factor. However, most of information regarding the utilisation of PEs was reported in slice distributions In addition, it is difficult to objectively measure the equivalence of the logic blocks from different FPGA vendors and technologies due to the relatively different internal architectures. Therefore, we use the approximate relationship between different FPGA slices done by [25]. Based on the report, the Xilinx Virtex-4 slice (with minor modifications) was used as a reference for all Virtex FPGA families before the Virtex-5 and all Spartan FPGA families. According to their universal method (based on experiments with real-world designs) one Virtex-5 slice is equivalent to 2 slices of the Xilinx Virtex-4 and one Altera ALM is equivalent to 1.3 slice of Virtex-4. By using this relationship, we normalised the slices used from works done by [17], [18] ,[19], [20], [21], [22], [23] and our proposed core. The normalised slice is again normalised to their corresponding number of logic cells used for all PEs reported in literature. Table 3 summarised the normalised CUPS performance on different types of FPGA. Due to the limited information gathered in literature, the normalised CUPS/logic cell measurement of several FPGA implementations [18], [20], [21] and [23] could not be effectively compared.

Table 3. Normalised CUPS performance

Target Device	Reported CUPS (Giga)	Normalised CUPS/LC (Mega)	Normalised CUPS/LC /Process Technology
XC2V6000 [17]	7.66	0.30	1.16E-04
XC2V6000 [19]	7.60	0.13	5.10E-05
EP2S180 [22]	25.6	0.01	7.34E-06
Proposed	39.00	1.73	1.56E-04

The normalised CUPS/logic cell/process technology (the last column in Table 3) is another proposed normalised performance metric (albeit test with speed). This new metric is calculated by multiplying the normalised CUPS per logic cell with their corresponding FPGA's LUT propagation delay. After taking out the advantages of virtex-5 FPGA by introducing these new metrics, our FPGA core performance remains the fastest and hence outperforms other implementations.

7. Conclusions and future work

In this paper, we have presented the processing element's design for the optimal alignment algorithms and its implementation on reconfigurable hardware platform. We evaluate the performance of our proposed hardware acceleration against GPU implementation as well as the SSEARCH35 and previous FPGA

implementations from two different performance perspectives; speed up and CUPS. New metrics are also proposed as independent performance evaluators to compare our core with others. The normalised performance evaluators showed that the proposed architecture remains the fastest. For longer query sequences, this architecture supports a multi-pass architecture with *n*-folding factor depending on logic densities available inside the target FPGA. Future work includes implementing overlapped computation and loading of the configuration data into the core for greater performance.

8. References

[1] O. Storaasli and D.Strenski, "Exceeding 100X Speedup/FPGA Cray XD1 Timing Analysis Yields Further Gains.," presented at Cray Users Group (CUG) Annual Technical Conference, CUG 2009, Atlanta, Georgia, 2009.

[2] D. A. Benson, et al., "GenBank," *Nucleic Acids Research*, vol. 28, pp. 15-18, 2000.

[3] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of Molecular Biology*, vol. 215, pp. 403-410, 1990.

[4] W. R. Pearson and D. J. Lipman, "Improved Tools for Biological Sequence Comparison," presented at National Academy of Sciences of the United States of America, 1988.

[5] M. Farrar, "Striped Smith-Waterman speeds database searches six times over other SIMD implementations," *Journal of Bioinformatics*, vol. 23, pp. 156-161, 2007.

[6] O. Storaasli, W. Yu, D. Strenski, and J. Maltby, "Performance Evaluation of FPGA-Based Biological Applications," presented at Cray Users Group (CUG) Annual Technical Conference, CUG 2007, Seattle, Washington, USA 2007.

[7] A. Boukerche, A. C. M. A. de Melo, and M. Ayala-Rincon, "Parallel strategies for local biological sequence alignment in a cluster of workstations," presented at 19th IEEE International Conference on Parallel and Distributed Processing Symposium, 2005.

[8] O. Tim, S. Bertil, and M. Douglas, "Hyper customized processors for bio-sequence database scanning on FPGAs," presented at 13th international symposium on Field-programmable gate arrays (ACM/SIGDA), Monterey, California, USA, 2005.

[9] H. T. Kung, "Systolic (VLSI) arrays for relational database operations," in *international conference on Management of data (SIGMOD '80)* New York, USA, 1980, pp. 105-116.

[10] R. Durbin, Eddy, S., Krogh, A., Mitchison, G, *Biological Sequence Analysis: Probabilistic Models for Proteins and Nucleic Acids*: Cambridge University Press, Cambridge UK, 1998.

[11] E. SR., "What Is Dynamic Programming?," in *Magazine of Nature Biotechnology*, vol. 22, 2004, pp. 909-910.

[12] B. Schmidt, H. Schroder, and M. Schimmler, "Massively parallel solutions for molecular sequence analysis," presented at International Symposium on Parallel and Distributed Processing (IPDPS 2002), 2002.

[13] M. Borah, R. S. Bajwa, S. Hannenhalli, and M. J. Irwin, "A SIMD solution to the sequence comparison problem on the MGAP," presented at International Conference on Application Specific Array Processors, 1994.

[14] P. Guerdoux-Jamet and D. Lavenier, "SAMBA: hardware accelerator for biological sequence comparison," *Journal of Computer applications in the biosciences : CABIOS*, vol. 13, pp. 609-615, 1997.

[15] Y. Yamaguchi, Maruyama, T., and Konagaya,A., "High Speed Homology Search with FPGAs," in *The Pacific Symposium on Biocomputing*, 2002, pp. 271-282.

[16] T. Van Court and M. C. Herbordt, "Families of FPGA-based algorithms for approximate string matching," presented at 15th IEEE International Conference on Application-Specific Systems, Architectures and Processors, 2004.

[17] K. Benkrid, L. Ying, and A. Benkrid, "A Highly Parameterized and Efficient FPGA-Based Skeleton for Pairwise Biological Sequence Alignment," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, pp. 561-570, 2009.

[18] X. Meng and V. Chaudhary, "Boosting data throughput for sequence database similarity searches on FPGAs using an adaptive buffering scheme," *Journal of Parallel Computing*, vol. 35, pp. 1-11, 2009.

[19] T. F. Oliver, B. Schmidt, and D. L. Maskell, "Reconfigurable architectures for bio-sequence database scanning on FPGAs," *IEEE Transactions on Circuits and Systems II*, vol. 52, pp. 851-855, 2005.

[20] X. Meng and V. Chaudhary, "A High-Performance Heterogeneous Computing Platform for Biological Sequence Analysis," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, pp. 1267-1280, 2010.

[21] A. Mohamed, E.-A. Esam, and T. Mohamed, "DNA and Protein Sequence Alignment with High Performance Reconfigurable Systems," in *the Second NASA/ESA Conference on Adaptive Hardware and Systems*: IEEE Computer Society, 2007.

[22] Z. Peiheng, T. Guangming, and R. G. Guang, "Implementation of the Smith-Waterman algorithm on a reconfigurable supercomputing platform," in *the 1st international workshop on High-performance reconfigurable computing technology and applications*. Reno, Nevada: ACM, 2007.

[23] J. Xianyang, L. Xinchun, X. Lin, Z. Peiheng, and S. Ninghui, "A Reconfigurable Accelerator for Smith Waterman Algorithm," *IEEE Transactions on Circuits and Systems II*, vol. 54, pp. 1077-1081, 2007.

[24] Xilinx, "Programmable Logic Design : Quick Start Handbook," Xilinx Inc., 2006.

[25] C. Technologies, "FPGA Logic Cells Comparison," Core Technologies, Russia, 24 Radio str., Moscow, Russia, 105005.

A Novel Efficient FPGA Architecture for HMMER Acceleration

M.Nazrin M.Isa, Khaled Benkrid, Thomas Clayton
System Level Integration Group,
School of Engineering,
University of Edinburgh,
EH9 3JL, Edinburgh
{m.n.isa,k.benkrid,t.clayton}@ed.ac.uk

***Abstract*—In this paper, a novel efficient FPGA-based architecture for the acceleration of the *hmmsearch* tool for biological sequence-to-profile alignment, which is based on the Viterbi algorithm, is presented. Typical hardware implementations of this Dynamic Programming-based algorithm require an amount of block RAMs proportional to the profile length in order to hold emission and transition probability scores for alignment matrix computation. In contrast, the proposed architecture uses the abundant logic slices available on FPGA as look-up tables or configuration elements (CEs) to hold the probability scores. Moreover, double buffering is used to efficiently manage a fixed number of CEs (equal to two i.e. CE_0 and CE_1) by scheduling both alignment matrix computation and processing element (PE) configuration to run simultaneously. This way, both time and space complexities are optimized, thus supporting multiple-pass or folded computation with significant throughput increases. In addition, with the fixed number of CEs, computational parameters such as number of folds and query profile's length could be changed at run time. Implementation results show that the core achieves 5.86 normalized speed-up per logic cell/process technology compared to the state-off-the-art.**

Keywords- Field Programmable Gate Array (FPGA), HMMER, hmmsearch, Sequence Alignment, Systolic Arrays, Viterbi Algorithm, Hidden Markov Models, Bioinformatics.

I. INTRODUCTION

Searching for sequence homology is a fundamental task in bioinformatics. The search enables discovery of invaluable clues to the function of genes and their evolutionary relationships. This leads to a plethora of applications including identification of related sequences in other living organisms, constructions of phylogenetic trees, and drug engineering. *HMMER* is one of the most widely used software tools for sequence homology. The main elements for this Hidden Markov-based sequence alignment package are *hmmsearch* and *hmmpfam*. The former searches for a profile Hidden Markov Model (HMM) in a sequence database, while the latter searches for one or more sequences in profile HMMs database. HMMs has been used in speech recognition for years and they are adopted in molecular biology due to their ability to search for low identity and distant sequences [1]. A profile HMM is essentially a probabilistic model that represents position specific highly conserved sequence patterns or motifs as a result of multiple sequence alignment. Motifs exist in evolutionary-related sequences and variations from common ancestors are due to the processes of mutation, selection, and genetic drift. These manifest themselves as residue substitution, deletion or insertion.

A profile HMM is modeled by discrete states, whereby each state represents motif positions with probability scores assigned to the state and its transitions. To understand this representation, it is worth to imagine that an HMM generates a sequence [2]. When a state is visited, a residue is emitted from the state based on emission probability score. On the other hand, transition to the next state depends on the state with the highest transition probability score. Transition from state to state generates the underlying state path referred to as a Markov chain.

Fig. 1 illustrates a profile HMM with full plan 7 architecture that is used in *HMMER*. The model has four sets of match (*M*), insertion (*I*) and deletion (*D*) states. Each *M* state represents one consensus column and a set of *M, I, D* states is the main element of the model and is referred to as a "node" in HMMER. The insertion state is a self-transition state and multiple insertions could occur between consensus columns. State B (begin) and E (end) are the flanking states of the main model and they are non-emitting states. The other states i.e. *S*, *N*, *C*, *T* and *J* are special states. Both flanking and special states control algorithm dependent features of the model, such as alignments with local or multiple-hit [3]. Alignment with multiple-hit occurs if the feedback score from state *J* is larger than state *N*. However, this case occurs very rarely and if it does, the search sequence usually comes from the family of the query profile HMM. In general, for a profile HMM of length L_m motif positions, the plan 7 HMM model shall comprise of L_m sets of *M, I, D* states, a set of flanking states and a set of special states. Note that, in any profile HMM length, there is no deletion state for the first set as well as deletion and insertion states for the last set. Given a sequence and a profile HMM, there are potentially many state paths to generate the same sequence. Only the path with the highest probability score will be chosen and this is dictated by the efficient DP-based Viterbi algorithm (the pseudo code is shown in Fig 2). The inner loop of the code comprises of three two dimensional matrices (*M, I, D*), which calculate scores of all motif positions involved in the main models for each of the residue. The outer loop consists of flanking and special states. They are calculated at the last motif position of the query profile.

In the following section, related work on FPGA-based HMMER acceleration will be discussed. Section III describes fine-grain parallelism offered by systolic arrays to accelerate the Dynamic Programming (DP)-based Viterbi algorithm in hardware. The discussion continues with an explanation of the folding technique in sequence alignment to cope with query profiles that are longer than what is physically implementable on a particular FPGA chip. In section IV, a novel efficient scheduling strategy and architecture is presented. Section V details the case of recalculation due to erroneous speculative computation, and its probability of occurrence. The corresponding

hardware architecture of the complete *hmmsearch* acceleration is then presented in section V before conclusion and future works are laid out in the last section.

II. Related work

In this section, a brief review of FPGA-based

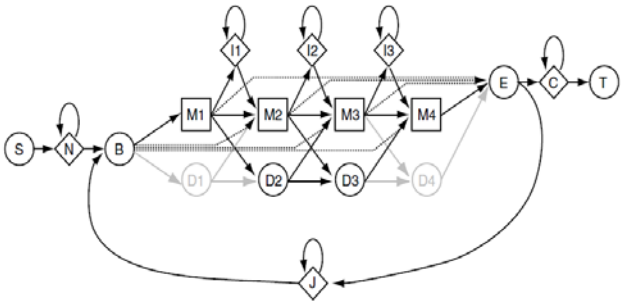


Figure 1. The Plan 7 Architecture[3]

For every sequence residue *i* from 0 to *n-1*

$$N(i) = N(i-1) + tr(N,N)$$

$$B(i) = \max\begin{cases} N(i) + tr(N,B) \\ J(i-1) + tr(J,B) \end{cases}$$

$$M(i,0) = I(i,0) = D(i,0) = -\infty$$

For every model position *j* from 0 to *m-1*

$$M(0,j) = I(0,j) = D(0,j) = -\infty$$

$$M(i,j) = e(M_j, S[i]) + \max\begin{cases} M(i-1,j-1) + tr(M_{j-1}, M_j) \\ I(i-1,j-1) + tr(I_{j-1}, M_j) \\ D(i-1,j-1) + tr(D_{j-1}, M_j) \\ B(i) + tr(B, M_j) \end{cases}$$

$$I(i,j) = e(I_j, S[i]) + \max\begin{cases} M(i-1,j) + tr(M_j, I_j) \\ I(i-1,j) + tr(I_j, I_j) \end{cases}$$

$$D(i,j) = \max\begin{cases} M(i,j-1) + tr(M_{j-1}, D_j) \\ I(i,j-1) + tr(D_{j-1}, D_j) \end{cases}$$

End

$$E(i) = \max\{M(i,j) + tr(M_j, E)\} \quad (j = 0,....,L_m - 1)$$

$$J(i) = \max\begin{cases} J(j-1) + tr(J,J) \\ E(i) + tr(E,J) \end{cases}$$

$$C(i) = \max\begin{cases} C(j-1) + tr(C,C) \\ E(i) \end{cases}$$

End

$$T(S,M) = C(N) + tr(C,T)$$

Figure 2. Pseudo Code of The Viterbi Algorithmn

hmmsearch implementation is discussed. This computation-intensive CPU program needs acceleration due to the exponential growth of protein families and sequence database sizes. Among early –reported work on FPGA-based *HMMER* acceleration we can find the work presented in [4] and [5]. These simplified the full plan 7 architecture (see Fig. 1) by neglecting the feedback loop *J*, to enables maximum parallelism of the systolic arrays. However, alignment without dependency of *J* state leaves no multiple-hit detection, which results in less accurate alignment scores especially for sequences that are closely related to the query profile. Other reported FPGA implementations that did not consider the feedback loop for alignment include [6], [7], and [8]. Although considering the *J* state guarantees accurate alignment scores, it requires quadratic time complexity. This is because only one cell is calculated per processing step. This impedes the parallel anti-diagonal computation of the Viterbi algorithm in systolic arrays. Fortunately, the probability of multiple-hit alignment is very low. Full parallelism of the time consuming algorithm could hence be speculatively calculated as reported in [9], [10], [11], [12], [13] and [14].

Typically, computing alignment scores in systolic arrays requires one processing element (PE) per profile HMM's node, where a PE implementation on a Xilinx Virtex-4 FPGA, for instance, consumes ~500 logic slices to implement the Viterbi algorithm. With profile HMM of average length ~200 nodes [15], about 100K logic slices and a large amount of block RAMs (BRAM) are required. Therefore, the folding technique has been reported in the literature to allow for longer profile HMM implementations on arbitrary-sized FPGA chips. This technique reuses PEs for computing alignment scores through several passes. For instance, in a linear systolic array of size *nPE* and a profile HMMs of length L_m, where L_m > *nPE*, a fold factor of $F=L_m/nPE$ is required. Through folded architecture, the alignment is performed in *F* passes over the same systolic arrays of size *nPE*. For subsequent processing passes, the PE must be updated with new emission and transition scores (henceforth referred to as coefficients) before alignment computation starts. In addition, a feedback FIFO (*First-In-First-Out*) is required to temporarily store intermediate data between passes. Previous work on FPGA-based HMMER acceleration has seen the use of Block RAMs (BRAMs) to hold coefficients for alignment matrix computation. In terms of area utilization, the configuration chain requires a proportional amount of BRAMs as the number of PEs increase. In addition, computing alignment matrix in multiple-pass requires the serial configuration chain to update all PEs with coefficients for every fold computation. This increases PE configuration time by a factor of *F*, where *F* is the number of fold.

Alternatively, we propose a hardware architecture with a fixed number of CEs (equal to 2) to hold coefficients for alignment matrix computation in each PE instead of replicating configuration data in a proportion with the folding factor. The CEs can be implemented using abundant FPGA logic slices. This enables the efficient use of the less abundant BRAMs for other tasks. In addition, an efficient scheduling strategy between alignment matrix computation and CE configuration is implemented into the core to effectively manage the fixed number of CEs. This optimizes area (logic and memory) resources and overall time complexity. Another attractive feature of our core is that computational parameters such as number of folds and input profile HMMs could be changed at run time.

III. Parallelizing the Viterbi algorithm and Processing It In Multiple-Pass Computation.

Systolic array is a widely used technique to harvest parallelism offered by FPGA. In the DP-based Viterbi algorithm, the recursive equation is divided into smaller sub-problems and these sub-problems are computed in parallel using systolic arrays. Given L_m the length of protein family and L_s the length of subject sequence, computing this recursive equation using a linear systolic

array of size L_m x L_s results in $O(L_m + L_s\text{-}1)$ time complexity. This is due to advantage of the anti-diagonal computation as indicated by the dotted lines in Fig. 3. In this diagram, a fold factor of four is assumed, i.e we want to align a motif position of length $4N_{PE}$ and we can only implement N_{PE} in hardware. This results in computation of L_m=16 in four different passes (F=4). During each pass, the intermediate results are stored in a FIFO for subsequent fold computation.

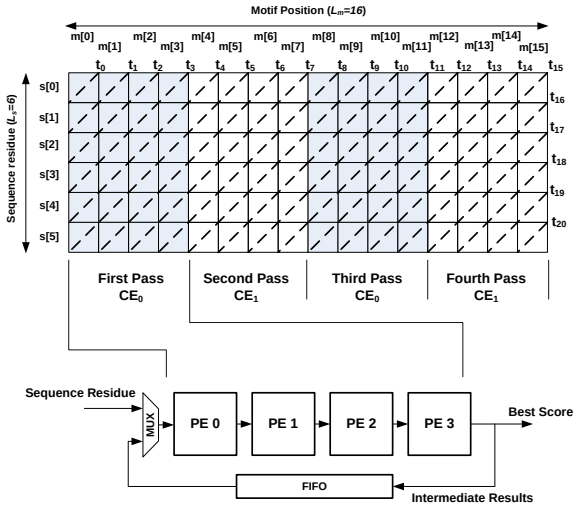


Figure 3. Parallelizing the DP Algorithm in Multiple-pass computation

IV. The Efficient Scheduling Strategy

This strategy is based on the widely used ping-pong or double buffering technique. In this paper, it is referred to as overlapped computation and configuration (OCC), whereby the operation to compute the alignment matrix is overlapped with CE configuration as illustrated in Fig. 4. This way, the configuration time is virtually removed, thus optimizes total execution time of the DP-based Viterbi algorithm. To allow for efficient scheduling, initially, all CE_0 elements in the pipeline will be configured with coefficients during *Initial Config.* phase. This is the only non-overlapping configuration operation. Once the first pass (F_1) computation starts, CE_1 in the pipeline will be updated with new coefficients for subsequent fold computation (labeled as *Overlapped 1* in Fig.4). This overlapped operation continues until all subject sequences in the database are exhausted. Note that, during *Overlapped 4*, CE_0 is updated with new coefficients for the next subject sequence.

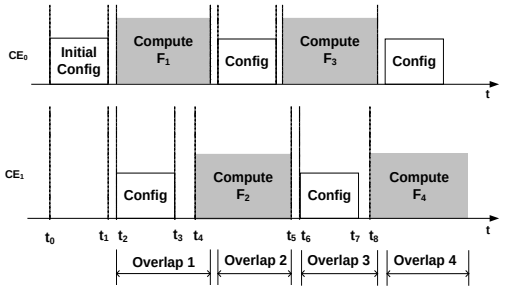


Figure 4. Efficient Scheduling Strategy between Alignment Matrix Computation and CE Configuration

V. System Architecture

In this section, the entire hardware design of a general purpose dynamic programming based algorithm (mentioned in section I) is presented. The main instance of this *hmmsearch* accelerator is the *PE_BLOCK*, which accelerates the DP-algorithm using a linear systolic array. It comprises of a pipeline of basic processing elements (PE_i). Each PE has two configuration elements (CE_0 and CE_1). A CE is essentially made up of FPGA logic slices and it consists of three look-up tables; 1) 20 emission scores of *M* state 2) 20 emission scores of *I* state and 3) 9 transition state scores. Each CE, with *CEDepth* of 49 elements (total depth of all three look-up tables) represents a particular motif position. The mapping of a CE with its corresponding motif position is determined by the *CE-MOTIF MAPPER* as illustrated in Fig. 5. The instance allocates all CE_0 elements in the pipeline to hold coefficients during even-numbered pass computations, whereas CE_1 for all odd-numbered pass computations. During multiple-pass processing, the *FEEDBACK FIFO* temporarily stores intermediate results from each pass before they are fed back to PE_0 through the input multiplexer. The FIFO depth is dictated by the length of the subject sequence and it is set to the maximum. The *BEST SCORE FIFO* stores the highest score of each subject sequence. Note that, a special unit (*Recalc.Unit*) is designed inside the core. This instance monitors feedback score of every sequence residue at the last motif position and triggers the core into recalculation mode whenever score from the feedback loop is dominant i.e the case of recalculation. A *Recalculation FIFO* temporarily stores all *M(i,j)*, *I(i,j)* and *D(i,j)* scores of each PE, whereby the input from PEs will be selected by an N_{PE} *to 1* multiplexer. The *MAIN CONTROLLER* is a scheduler for the OCC operation. It manages the two CEs by alternately use them for computation and configuration depending on the aforementioned computation passes. For instance, while CE_0 holds the coefficients for alignment matrix computation, CE_1 will be configured with new coefficients for subsequent fold computation, and vice versa. This way, both computation and configuration modes run simultaneously, thus optimizes total execution time by virtually remove the configuration overheads as a result of the overlapped operation. In addition, the same systolic array (*PE_BLOCK*) can be reused for multiple-pass alignment matrix computation without incurring additional logic resources

Another important instance to implement the efficient scheduling strategy is the *CE LOADER*. It has two independent configuration chains i.e LCE_0 and LCE_1 where each of them is directly connected to CE_0 and CE_1 respectively in the pipeline PEs (see Fig. 5). This enables each CE to be updated independently with new coefficients whenever the pipeline has finished computation. The configuration chain is made up of circular buffers, which is efficiently implemented using shift registers look-up table (SRL) available in FPGA's slices. These buffers constantly cycle all coefficients of profile HMM, presenting complete columns of the corresponding motif positions at every multiple of CE_{Depth} clock cycles to the PEs. This way, all CEs are configured simultaneously, with a worst case configuration time of $2xCE_{Depth}$ clock cycles i.e $tCE_x \leq 2 \times CE_{Depth}$. The total configuration time, taking

into account the loader configuration time is defined by (1),

$$T_{config} = tLCE_x + tCE_x \quad (1)$$

Where, $tLCE_x = N_{PE} \times CE_{Depth}$. It is the time to update the configuration chain with new coefficients for the case of aligning longer profile HMMs in multiple-pass computation.

Fig. 6 illustrates internal structure of the processing element following the pseudo code shown in Fig. 2. It implements elementary operations of the Viterbi algorithm i.e the two dimensional matrices *M(i,j)*, *I(i,j)* and *D(i,j)*. The score of each matrix is calculated in parallel and their output is delayed by one clock cycle i.e *M(i-1,j-1)*, *I(i-1,j-1)* and *D (i-1,j-1)*. To illustrate the data dependencies between PEs, consider PE_2 in Fig. 3 as an example, whereby the systolic operation computes alignment score of residue *s[1]* at *t3*. These data dependencies require outputs of previous PE (i.e. PE_1) from t_1 and t_2 for left and upper-left dependencies of the PE respectively. The PE also requires its own output from previous processing step, t_2. The left, upper and upper-left dependencies of *M*, *I* and *D* cells are then fed into PE_2 for its alignment score computation. The *E(i,j)* instance computes scores of *E* state, either from the most probable path that arrives at state *E* with transition from state *M* or the score of the path that ends at state E with self-transition. It compares the maximum score from the PE_{i-1} with the current *E(i,j)* score before emits the maximum of the two to PE_{i+1}. During each processing step the input residue *s[i]* is propagated to subsequent PE along with the *M*, *I*, *D* and *E* scores. Ultimately, the score from *E* emitted by the last PE in the final processing step is the score of the alignment. Other one dimensional matrices (*N*, *B*, *J*, *C* and *T*) are not implemented here to reduce the size of the PE. Note that both *dw* (data width) and *cdw* (compute data width) are parameterizable and in this core it is set as 5-bit and 15-bit respectively.

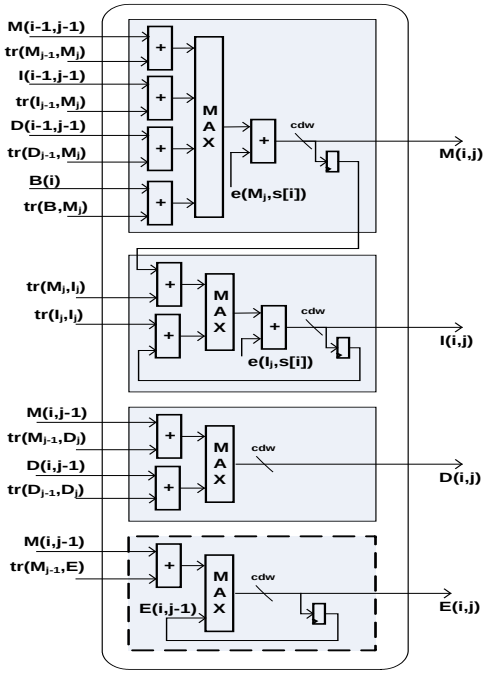


Figure 6. Internal PE Architecture with fixed CE (CE_0 and CE_1)

VI. THE CASE OF RECALCULATION

This section discusses the case of recalculation for the full plan 7 HMM architecture. We start this section by evaluating the significant of *J* score in an alignment. Several samples of randomly picked profile HMMs from *pfam* database [15] are searched against 84,479,584 residues of the *SwissProt* [15] protein knowledgebase. Based on the analysis, we found that chances for the feedback score to be selected i.e recalculation are ~0.01 percent, which is very low. Therefore, in most cases, the Viterbi algorithm could be calculated speculatively in parallel by eliminating dependency of *J* state for subsequent residue. This enables full parallelism of alignment matrix computation. Recalculation is considered when necessary and in worst case, it may occur at L_s*-1* times. In this case, computation turns to sequential fashion.

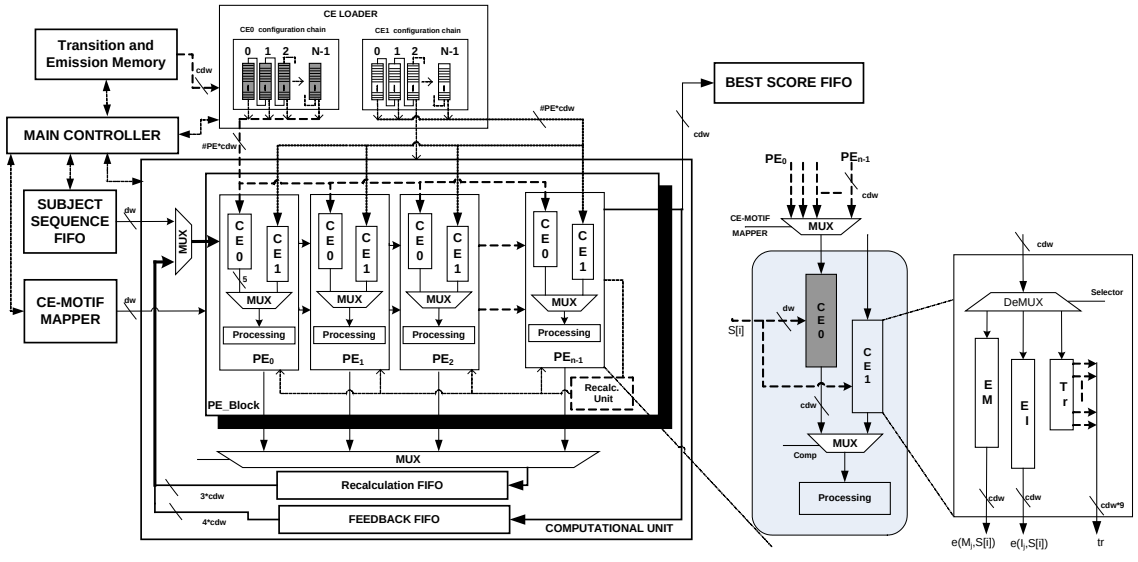


Figure 5. The overall system architecture with fixed configuration elements.

Fig. 7 illustrates the event of recalculation. In this diagram, a profile HMM has a length L_m=8 positions and a subject sequence length L_s =6 residues.

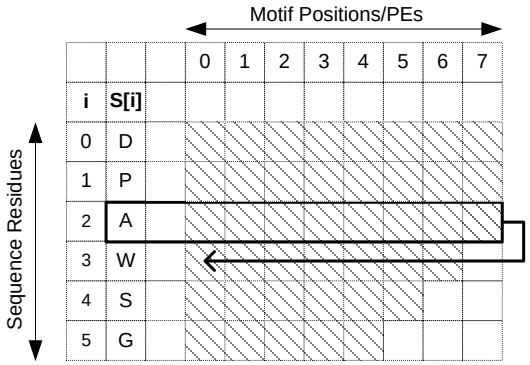


Figure 7. The case of recalculation

Initially, the alignment matrix is speculatively calculated. As residue '*A*' enters the last motif position at PE_7 of the profile HMM (assuming the number of PE equals to the profile's length), the *J* score is larger than *N(i-1) +tr(N,N)*, causing alignment of subsequent residue i.e '*W*' with feedback score. Consequently, all PEs beginning from PE_0 recalculate their new *M(i,j), I(i,j), D(i,j)* scores. The recalculation starts from row '*W*' with boundary values are taken from all previously computed *M(i,j), I(i,j), D(i,j)* scores of residue '*A*', which is stored in the *Recalculation FIFO*.

VII. Performance analysis

This section presents implementation results of the HMMER acceleration on FPGAs using the full plan 7 architecture. Our core is designed using Verilog HDL (Hardware Description Language) and implemented on Alpha Data ADM-XRC-5LX board with Virtex 5 FPGA (XC5VLX110) on it. A PE, which implements the Viterbi algorithm, utilizes ~337 logic slices, while each CE uses ~112 logic slices. Consequently, with 17,280 slices on the silicon chip, we are able to fit ~38 PEs. Moreover, the processing word length of this core is 15-bit and it is clocked at 166 MHz. Table I shows our core performance against various FPGA implementations measured in CUPS (Cell update per second). The CUPS figure is a common performance indicator, which is widely used in computational biology. It is determined by multiplying the number of PEs with the core's operating frequency. From Table I, it is clearly shows that the proposed core is the fastest in terms of operating frequency i.e 41 percent higher clock frequency than [11] and in terms of CUPS performance, it is the second fastest after [11]. This is because total amount of logic slice available in [11] is twice (after taking into account one Virtex 5 slice equals to two Virtex 4 slices) as compared to XC5VLX110 [16].

TABLE II. Normalized Speed Up Performance

Speed-up	Area ratio	speed up/area	LUT Delay Ratio	Normalized Speed-up
0.58	0.05	11.1	0.53	5.86

TABLE I. CUPS performance of the Full Plan 7

Ref	**Device**	**One PE (Slices)**	**#PEs**	**Freq (MHz)**	**CUPS Giga**
[9]	XC3S4000	~583	32	60.0	1.92
[10]	XC5VLX110	-	25	130.0	3.20
[11]	XC4VLX160	342	100	117.9	11.80
[12]	XC2V6000	451	30	70.0	2.10
[13]	XC3S1500	451	10	70.0	0.70
[14]	XC3S1500	451	10	70.0	0.70
ours	XC5VLX110	337	38	166.0	6.30

Although the CUPS figure is widely used to evaluate performance of systolic arrays, it depends on the number of PEs, which varies depending on types of FPGA in hand. Any silicon chip with higher slices could implement more PE resulting in higher CUPS figure and vice versa. Therefore, another performance indicator; speed-up which measures all overhead time including pipelining filling/flushing as well as other FPGA communication overheads is used to effectively evaluate the core performance. In an attempt to fairly evaluate our core performance against others, we normalize the speed-up with respect to area (logic and memory) and process technology. The normalized performance indicator is shown in (2).

$$SpeedUp_{normalized} = \frac{SpeedUp}{LC_{equ}} \times LUT_{Delay} \quad (2)$$

This will taking out the advantages of Virtex 5 FPGA against it predecessors. Where LC_{equ} is the total amount of logic cells (logic and memory resources) used to infer PEs. LUT_{Delay} is the FPGA's slice Look-up Table (LUT) delay. Due to different internal slice architecture in FPGAs, LC will be used in this analysis rather than logic slice. The former is an abstract logic resource measure independent of any particular FPGA family's slice architectures. For fair and meaningful comparison, we align the same profile HMMs (*Pkinase*, L_m=294) against a sequence (*Artemia*, L_s = 1405) as reported in [11]. Our core requires 8 folds and its total execution time is 97.18 us (103x and 3.28x speed-up compared to HMMER2 and HMMER3 respectively). Ref. [11] with more slices on chip only requires 3 folds to compute the same model, resulting in less execution time (56.0 us). By dividing execution time of [11] with our's, the core speed-up is 0.58. This is due to the slow down factor of multiple fold computations.

To evaluate them independent of FPGA devices and process technology, we normalize the speed-up performance per logic cell and process technology. Note that one Virtex 5 logic slice has twice as many slices as it predecessors. This is due to the slightly more complex architecture of SLICEL and SLICEM of Virtex 5 families as compared to previous Xilinx FPGA families. An LC in Xilinx FPGA comprises of a look up table (LUT), a multiplexer and a register. The LUT can also be used as distributed RAM or as a shift register [17]. Using this definition and information provided in [18], equivalent LCs in each slice of Virtex 5 and it predecessors is calculated. From the analysis, we found that one slice of Virtex 5 FPGA equals to four LCs and one slice of previous Xilinx FPGA families only has two LCs. Since our PE has no BRAM element in it, then the equivalent

logic cells/PE is 1,348 LCs. For the other FPGA implementations, which utilizes BRAM in the PE to hold coefficients, equivalent amount of LCs has to be considered prior to normalization. This is done by synthesizing a FIFO and a PE using both the Cadence Build Gates (2005) with 0.18um UMC process technology and Xilinx ISE 13.1 targeting two different Virtex architectures (XC4VLX160 and XC5VLX110) and the equivalent gate count of each is noted. This allows us to normalize the speed-up figure of all PEs (logic and memory) in terms of LCs. In the case of [11], one PE utilizes ~18Kbit BRAM or equivalent to 8,622 LCs (1 Kbit memory equals to 479 LCs for XC4VLX160 FPGA). After taking the memory element into account, the speed-up is normalized and its corresponding performance is shown in table II. Speed-up is calculated by dividing the execution time of [11] with ours, while the area ratio is determined by dividing the total LCs to infer 100 PEs in [11] with 38 PEs in our proposed core. By dividing the calculated speed-up with the area ratio, we found that the area normalized speed-up is 11.1. To take out the advantages of the Virtex 5 FPGA in terms of fabrication technology, we then multiply the normalized figure with the LUT propagation delay ratio. This gives the normalized speed-up per LC/process technology of 5.86, which clearly shows that our proposed core outperforms [11] after normalization. Due to limited information provided in the literature, the normalized performance indicator cannot be used to fairly evaluate other reported implementations.

VIII. CONCLUSION

In this paper, we presented a novel efficient FPGA architecture for *hmmsearch* acceleration. In particular, a double buffering based technique is devised to optimize both time and space complexities of the Viterbi algorithm based implementation resulting in the use of only two configuration elements per processing element to hold transition and emission probability scores for any profile HMM length. Indeed, by overlapping computation and configuration, this technique enables the implementation of arbitrary L_m length HMMs on any particular FPGA chip using folding technique and multiple-passes computation with no dependency on BlockRAM resources, all at high performance. Implementation results show the proposed architecture achieves a normalized speed-up of 5.86 compared to state-off-the-art FPGA implementations. Future work includes scaling current design on a multi-FPGA computer.

REFERENCES

[1] R. Durbin, Eddy, S., Krogh, A., Mitchison, G, *Biological Sequence Analysis: Probabilistic Models for Proteins and Nucleic Acids*: Cambridge University Press, Cambridge UK, 1998.

[2] S.R.Eddy, "What is Hidden Markov Model?," *Computational Biology*, vol. 22, 2004.

[3] S.R.Eddy, "HMMER User's Guide," Washington University School of Medicine 2003.

[4] T. Oliver, et al., "Accelerating the Viterbi Algorithm for Profile Hidden Markov Models Using Reconfigurable Hardware Computational Science – ICCS 2006," vol. 3991, *Lecture Notes in Computer Science*: Springer Berlin / Heidelberg, 2006, pp. 522-529.

[5] P. M. Rahul, B. Jeremy, D. C. Roger, A. F. Mark, and H. Brandon, "Accelerator design for protein sequence HMM search," in *Proceedings of the 20th annual international conference on Supercomputing*. Cairns, Queensland, Australia: ACM, 2006.

[6] A. C. Jacob, J. M. Lancaster, J. D. Buhler, and R. D. Chamberlain, "Preliminary results in accelerating profile HMM search on FPGAs," presented at Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International, 2007.

[7] K. Benkrid, P. Velentzas, and S. Kasap, "A High Performance Reconfigurable Core for Motif Searching Using Profile HMM," presented at Adaptive Hardware and Systems, 2008. AHS '08. NASA/ESA Conference on, 2008.

[8] T. F. Oliver, B. Schmidt, Y. Jakop, and D. L. Maskell, "High Speed Biological Sequence Analysis With Hidden Markov Models on Reconfigurable Platforms," *IEEE Transactions on Information Technology in Biomedicine*, vol. 13, pp. 740-746, 2009.

[9] D. Steven and Q. Patrice, "Hardware Acceleration of HMMER on FPGAs," *J. Signal Process. Syst.*, vol. 58, pp. 53-67, 2010.

[10] S. Yanteng, et al., "Accelerating HMMer on FPGAs using systolic array based architecture," presented at Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on, 2009.

[11] T. Takagi and T. Maruyama, "Accelerating HMMER search using FPGA," presented at International Conference on Field Programmable Logic and Applications, 2009. FPL 2009. , 2009.

[12] T. Oliver, L. Y. Yeow, and B. Schmidt, "Integrating FPGA acceleration into HMMer," *Parallel Computing*, vol. 34, pp. 681-691, 2008.

[13] O. Tim, "High Performance Database Searching with HMMer on FPGAs," 2007.

[14] W. John Paul, et al., "MPI-HMMER-Boost: Distributed FPGA Acceleration," *J. VLSI Signal Process. Syst.*, vol. 48, pp. 223-238, 2007.

[15] pfam, "pfam database," 2009.

[16] Xilinx, "Virtex-5 Family Overview," Xilinx Inc DS110, 2009.

[17] C. M. Maxfield, *The Design Warrior's Guide to FPGAs*: Elsevier, 2004.

[18] A. Percey, "Advantages of the Virtex-5 FPGA 6-Input LUT Architecture," Xilinx Inc. WP284, 2007.

A Highly Efficient Substitution Matrix Loader for Pairwise Sequence Alignment

M.Nazrin Md.Isa, Khaled Benkrid, Thomas Clayton
System Level Integration Group,
School of Engineering,
University of Edinburgh,
EH9 3JL, Edinburgh
{m.n.isa,k.benkrid,t.clayton}@ed.ac.uk

***Abstract*—This paper presents a novel substitution matrix loader architecture for pairwise sequence alignment. The search for sequence homology using DP-based alignment matrix computation is an important tool in molecular biology. It can be implemented either by optimal or sub-optimal approaches. Both of these methods require frequent and rapid access to the residues probability scores for PE (Processing Element) configuration especially in a folded systolic array. Typical FPGA implementations configure look-up tables in the pipeline PEs either by using a serial configuration chain with different look-up tables or by run time reconfiguration of the same look-up table. In the former case, configuration time increases proportionally to the number of look-up tables, while the latter case suffers from the limited reconfiguration bandwidth. Therefore, in this paper, we propose a highly efficient parallel loader to optimize both time and space complexities of protein sequence alignment in folded systolic arrays, using only two configuration elements (CEs). In addition, the proposed loader enables PEs to be updated with substitution matrix scores concurrently, with the worst case configuration time of 2 x the depth of the PE's look-up table (in clock cycles). This allows for further optimization of the most time consuming alignment matrix computation through efficient scheduling of alignment matrix computation and PE configuration. Implementation results show that the proposed architecture achieves $k.N_{PE}$ speed-up in configuration time (where k is the folding factor and N_{PE} is the number of PEs) compared to classical approaches, at virtually no area overhead.**

Keywords: Field Programmable Gate Arrays, Sequence Alignment, Folded Systolic Array, Substitution Matrix, Smith Waterman, Needleman-Wunsch .

I. INTRODUCTION

Sequence alignment is a fundamental tool in molecular biology with a plethora of applications including drug engineering, forensics and early disease diagnosis. Searching for sequence homology could be done either by optimal or sub-optimal search techniques. The former searches for optimal scores between database sequences and query sequences e.g. a newly discovered biological sequence. The search technique is usually based on dynamic programming (DP) e.g. the Smith Waterman and the Needleman-Wunsch algorithms. On the other hand, sub-optimal methods use heuristic approaches as a tradeoff between speed and sensitivity. Given a threshold score *T*, sequences with scores higher than *T* will be selected for full dynamic programming based alignment. In either case, aligning sequences in a realistic time becomes an issue due to the exponential growth of database sequences. Advancement in computing technologies has seen the use of parallel architectures such as FPGAs in [1], [2], [3], [4],[5] and GPUs in [6], [7], [8], [9] to accelerate the time consuming DP-based algorithm. In hardware, the algorithm is usually accelerated using a linear systolic array. The latter consists of an array of processing elements (PEs) with one PE holding one amino acid residue. For sequences longer than the maximum possible number of PEs on a particular chip (N_{PE}), the computation is performed by a folded systolic architecture with *k*-folds, whereby *k* is equal to the sequence length divided by N_{PE}. This way, PEs are reused between passes to complete a whole pairwise sequence alignment. Typical FPGA implementations configure the pipeline PEs with *k* fold factors by using a serial configuration chain, which updates probability scores in the PEs sequentially includes in [1], [2]. This requires each PE to have *k* different look-up tables and consequently, both configuration time and space complexities increase proportionally to $k.N_{PE}$. Another method uses run time reconfiguration (RTR) to configure a look-up table in the PE during alignment matrix computation [3]. Although, this approach optimizes space complexity, it suffers from the limited bandwidth of the configuration mechanism (e.g. the maximum bandwidth of the Internal Configuration Access Port, ICAP). Therefore, in this paper, we present an alternative way to optimize the configuration and computation architecture both in time and space complexities. A highly efficient parallel loader is proposed which updates all PEs simultaneously using only two configuration elements (CE) per PE.

The remainder of this paper is organized as follows. In the following section, important background on biological sequence alignment and a brief overview of an optimal algorithm are presented. Then, section III details the internal architecture and operation of the proposed loader. In section IV, the performance of the loader in terms of speed, area and power is discussed before conclusions are laid out.

II. BACKGROUND

Biological sequences diverge from their common ancestors due to the process of mutation, selection and random genetic drift [10]. For instance, mutation involves three main

processes: substitution of residues, insertion of new residues and deletion of existing residues. Both insertion and deletion are referred to as gaps. Gaps in alignments are penalized when scoring. The cost of gaps depends on its length and generally, there are two different ways to penalize gaps: linear and affine gap penalties. Equation (1) shows the Needleman-Wunsch algorithm with a linear gap penalty [10] as an example of optimal alignment algorithms which is used to compute the best global alignment (and score). In it, given a query sequence, $X = x_1,x_2,x_3..x_i...x_M$ (of length M) and $Y = y_1,y_2,y_3..y_j...y_N$ (of length N), this DP-based alignment algorithm searches for the best alignment between sub-sequences of x and y using an alignment matrix $F(i,j)$. This M *by* N matrix calculates the largest score among three alternatives in a recursive manner (see (1)). Here $s(x_i,y_j)$ represents the probability of substituting residue x_i for residue y_j according to a probability model stored in the form of a *substitution matrix* (see Fig. 1 for instance).

$$F(i,j) = \max\begin{cases} F(i-1,j-1) + s(x_i, y_j) \\ F(i-1,j) - d \\ F(i,j-1) - d \end{cases} \quad (1)$$

BLOSUM50, *BLOSUM62* and *PAM* are examples of substitution matrices [10], [11] and [12]. Each column and row in a substitution matrix represent individual amino acid residues with the intersection representing a score denoting the likelihood of substituting one residue for the other. Note that entries on the main diagonal are highlighted in bold, and represent scores of identical residue pairs.

	A	C	D	E	F	G	H	I	K	L	M	N	P	Q	R	S	T	V	W	Y
A	**5**	-1	-2	-1	-3	0	-2	-1	-1	-2	-1	-1	-1	-1	-2	1	0	0	-3	-2
C	-1	**13**	-4	-3	-2	-3	-3	-2	-3	-2	-2	-2	-4	-3	-4	-1	-1	-1	-5	-3
D	-2	-4	**8**	2	-5	-1	-1	-4	-1	-4	-4	2	-1	0	-2	0	-1	-4	-5	-3
E	-1	-3	2	**6**	-3	-3	0	-4	1	-3	-2	0	-1	2	0	-1	-1	-3	-3	-2
F	-3	-2	-5	-3	**8**	-4	-1	0	-4	1	0	-4	-4	-4	-3	-3	-2	-1	1	4
G	0	-3	-1	-3	-4	**8**	-2	-4	-2	-4	-3	0	-2	-2	-3	0	-2	-4	-3	-3
H	-2	-3	-1	0	-1	-2	**10**	-4	0	-3	-1	1	-2	1	0	-1	-2	-4	-3	2
I	-1	-2	-4	-4	0	-4	-4	**5**	-3	2	2	-3	-3	-3	-4	-3	-1	4	-3	-1
K	-1	-3	-1	1	-4	-2	0	-3	**6**	-3	-2	0	-1	2	3	0	-1	-3	-3	-2
L	-2	-2	-4	-3	1	-4	-3	2	-3	**5**	3	-4	-4	-2	-3	-3	-1	1	-2	-1
M	-1	-2	-4	-2	0	-3	-1	2	-2	3	**7**	-2	-3	0	-2	-2	-1	1	-1	0
N	-1	-2	2	0	-4	0	1	-3	0	-4	-2	**7**	-2	0	-1	1	0	-3	-4	-2
P	-1	-4	-1	-1	-4	-2	-2	-3	-1	-4	-3	-2	**10**	-1	-3	-1	-1	-3	-4	-3
Q	-1	-3	0	2	-4	-2	1	-3	2	-2	0	0	-1	**7**	1	0	-1	-3	-1	-1
R	-2	-4	-2	0	-3	-3	0	-4	3	-3	-2	-1	-3	1	**7**	-1	-1	-3	-3	-1
S	1	-1	0	-1	-3	0	-1	-3	0	-3	-2	1	-1	0	-1	**5**	2	-2	-4	-2
T	0	-1	-1	-1	-2	-2	-2	-1	-1	-1	-1	0	-1	-1	-1	2	**5**	0	-3	-2
V	0	-1	-4	-3	-1	-4	-4	4	-3	1	1	-3	-3	-3	-3	-2	0	**5**	-3	-1
W	-3	-5	-5	-3	1	-3	-3	-3	-3	-2	-1	-4	-4	-1	-3	-4	-3	-3	**15**	2
Y	-2	-3	-3	-2	4	-3	2	-1	-2	-1	0	-2	-3	-1	-1	-2	-2	-1	2	**8**

Figure 1. Blosum 50 (rearranged to alphabetical order), with 20 columns by 20 rows of different amino acid residues.

III. THE PROPOSED EFFICIENT LOADER

The main function of our proposed novel loader is to load all PEs of a pairwise sequence alignment array with their corresponding substitution matrix columns simultaneously. This allows for efficient data transfer as the configuration time significantly reduced to $1/k.N_{PE}$ compared to conventional serial configuration techniques. Consequently, a PE with only two configuration elements could be used for any folding factor, whereby as one CE is being used for alignment matrix computation, the other CE is being updated with probability scores for subsequent fold computation. This optimizes both time and space complexities of the DP-based algorithms in hardware. Fig. 2 illustrates the loader and its parallel configuration chain connected to a PE. The port description of the loader is given in Table I.

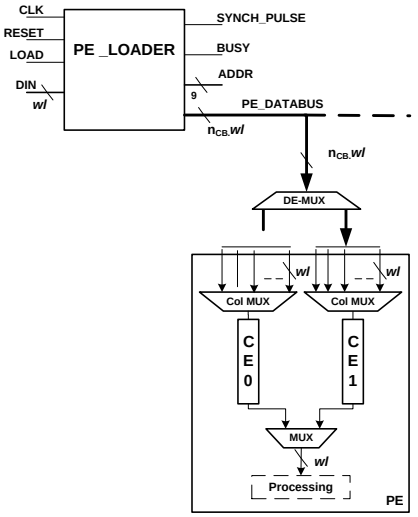


Figure 2. The parallel loader and the PE with two configuration elements (CE_0 and CE_1).

TABLE I. PORT DESCRIPTIONS OF THE EFFICIENT LOADER

Port	Description
CLK	Clock signal for synchronize the loader operations
RESET	When HIGH, it resets the loader to load different substitution matrix (if required)
LOAD	When HIGH, it triggers the loader to start shifting data into circular buffers, i.e. when substitution matrix memory is ready with probability scores
ADDR	Supply address to the substitution matrix memory to fetch its contents into the loader
DIN	Read out substitution matrix scores from the memory
PE_DATABUS	Data bus to update PEs with probably scores
BUSY	When HIGH, it indicates that the loader is shifting in substitution matrix scores into its circular buffers
SYNCH_ PULSE	A pulse signal whereby each pulse interval represents valid scores to configure the PEs

Fig. 3 illustrates the internal architecture of the loader with n_{CB} circular buffers to hold the columns of the substitution matrix e.g. n_{CB} = 20 for the substitution matrix of Fig. 1. The buffer has n_{row} shift registers, with each shifting one element of a particular substitution matrix row into the buffer, in turn, every clock cycle. The wordlength of the substitution matrix elements *wl* is parameterizable. For the case of Blosum50, 5-bit two's complement is enough to represent its elements, in which case the loader operates as 5-bit serial-in-serial-out shift register during initial configuration mode, and once in running mode, it operates as 5-bit serial in $n_{col} \times wl$-bit parallel out circular shift register. Details of these operations are presented in subsections A and B.

A. Initial Configuration Mode

Right shift operation is the fundamental operation of the loader during initial configuration mode. The operation starts by shifting elements of a given substitution matrix serially column by column into the corresponding buffers which are pipelined together in a long chain. Each element is shifted into the buffer chain every clock cycle. Consequently, all substitution matrix elements of one column are completely loaded into the buffer chain within n_{row} clock cycles. Note that, the thick broken line arrow in Fig. 3 depicts the flow of the shift operation during configuration mode. It begins to fill the last buffer i.e CB_{n-1} with the first n_{row} elements in the last column of the substitution matrix and continues with the following n_{row} elements to buffer $CB_{n-2.}$ This sequential shift operation continues until CB_0. This way, all scores will be loaded into the buffer chain according to their corresponding column. Once scores are completely loaded i.e. the memory read is finished, the loader is ready to configure the PEs. The initial configuration time (in clock cycles) to read an entire substitution matrix into the loader depends on the size of the substitution matrix and is mathematically expressed in (3).

$$t_{initload} = n_{col} \times n_{row} \quad (3)$$

Where, n_{col} is the number of columns and n_{row} is the number of rows of the substitution matrix. Details of the configuration mode function are described by the pseudo code in Fig. 4.

B. Running Mode

During this mode, all elements in a buffer are circulated every clock cycle following the direction of the arrow with the thin dotted line as shown in Fig. 3. Data circulation within the circular buffers ensures that valid scores are available for PE configuration within a maximum duration of $2.n_{row}$, as expressed in (4).

$$t_{config} \leq 2 \times n_{row} \quad (4)$$

This way, all PEs will be configured with probability scores concurrently with the worst case configuration time of $2 \times n_{row}$.

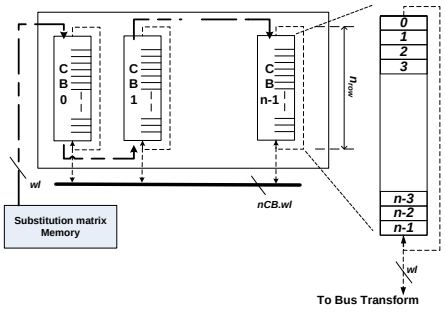


Figure 3. The Parallel Loader with the circular buffers

The loader architecture offers efficient way of fast configuration of the probability scores instead of updating PEs using a serial configuration chain as outlined in section I. Note that, the initial configuration mode runs only once whereas the loader remains in the running mode until it is reset to load other amino acid models.

Input: *Substitution matrix probability scores,* n_{CB}, n_{row}
CBcounter $\leftarrow n_{CB}$ *-1*
n $\leftarrow n_{row}$*-1*
For *every element of a given substitution matrix*
  If *(CBcounter!=0)*
    Shift element into the buffer chain
      Decrement CBcounter by one
      If *(n !=0)*
        Shift score into the corresponding row of the buffer
        Decrement n by one
      Else *reset n to* n_{row}*-1*
      End if
  Else
    Circulate scores inside their circular buffer
    Generate synch_pulses at each start of circulation
  End if
End for
Output: valid substitution matrix scores at synch_pulse intervals

Figure 4. Pseudo code for initial configuration mode and running mode.

C. Bus transformation

For the purpose of data transfer to the PE, a large bus is designed (see the bold horizontal line in Fig.3) to broadcast probability scores to all systolic array PEs. Indeed, all output ports of the circular buffers (*wl*-bit each) are joint together to form a large bus of width $n_{CB} \times wl$. The bus (*PE_DATABUS*) creation is shown in the Verilog pseudo code of Fig. 5.

For *every circular buffer beginning from n=20 until 0*
  Generate
    PE_DATABUS [n $\times$ *wl: (n-1)* $\times$ *wl] =*CB_{n-1}
  Endgenerate
Endfor

Figure 5. Pseudo code for a large bus creation

In the timing diagram shown in Fig. 6, the substitution matrix memory is assumed to be already filled with probability scores. During initial configuration mode, the loader operation is marked by the *BUSY* signal being HIGH. Once all scores are entirely loaded, the loader is ready for PE configuration, thus synchronization pulses (*SYNCH_PULSE)* are emitted every n_{row} clock cycles, whereby at each pulse interval, valid scores are available on the *PE_DATABUS* for PE configuration.

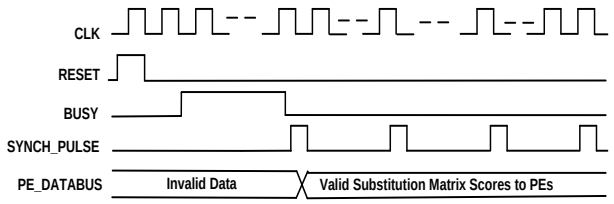


Figure 6. Valid Substitution matrix scores available to the PE during SYNCH_PULSE intervals.

PE configuration happens at any stage during this interval i.e. when its own probability scores are output by the circular buffers. Indeed, the query residue inside each PE selects its corresponding substitution matrix column (see multiplexer *Col MUX* in Fig 2).

IV. IMPLEMENTATION RESULTS

This section discusses the loader's implementation efficiency in terms of area, speed and power performance. The loader design was captured using Verilog HDL in a parameterizable manner. Two different CAD tools are used in this evaluation: Xilinx ISE 13.1 targeting a Xilinx XC5VLX110-3 FPGA and Cadence Build Gates version 2005 with 0.18um UMC process technology for ASIC implementation.

The operating frequency of the loader when synthesized on the XC5VLX110-3 FPGA (65nm based CMOS technology) is 396 MHz. Note that the loader's circular buffers are implemented efficiently on Xilinx FPGAs using the slices' LUT configuration referred to as SRL32[13]. Note also that typical systolic array FPGA implementations have PEs operating at a lower frequency e.g. 100-200 MHz. This means that the configuration time could be reduced by a factor of ~4 for higher performance if the loader is clocked separately. In terms of area utilization, the Xilinx ISE reported that the design utilizes 87 logic slices. We also synthesized the design using Cadence Build Gates with 0.18um UMC process technology. From the area report, we found that, the total area occupied by the loader was 142,980.92 um^2. The speed reported was 1GHz.

In an attempt to measure the power consumption of the design, we used a file generated from a map report (*sml.ncd*) to measure its static power and a simulation activity file (*sml.vcd*) to estimate the dynamic power of the loader. These files are then used as input parameters to the Xilinx ISE Xpower Analyzer. The power consumption of the loader was estimated to ~1.269W, with 1.144W as static power and 0.126W as dynamic power. The implementation results are summarized in Table II.

TABLE II. AREA,SPEED AND POWER PERFORMANCE

ASIC Implementation frequency (GHz)	1.0
ASIC Implementation Area (um^2)	142,980.92
FPGA logic slices (#slices)	87
FPGA Implementation Frequency (MHz)	396.1
Worst case Configuration time (ns)	101.0
Static Power (W)	1.14
Dynamic Power(W)	0.13

V. CONCLUSION

An efficient loader targeting the configuration of folded systolic arrays for pairwise sequence alignment was presented in this paper. The loader has an ability to configure the systolic array's processing elements in parallel with a worst case configuration time $2.n_{row}$ clock cycles where n_{row} is the number of rows of the substitution matrix. Unlike conventional substitution matrix loaders, which configure processing elements through a serial configuration chain, this loader enables concurrent PEs configuration regardless of the number of processing elements, using a fixed number of configuration elements (equal to 2). Implementation results show that the loader occupies a very small footprint (87 slices on a Virtex-5 FPGA) with a typical maximum clock frequency ~4 times faster than a typical systolic array operating frequency. As a result, this loader is able to optimize the time consuming configuration operation of optimal sequence alignment algorithms, especially in folded architectures, through the concurrent scheduling of alignment matrix computation and processing element configuration.

REFERENCE

[1] K. Benkrid, L. Ying, and A. Benkrid, "A Highly Parameterized and Efficient FPGA-Based Skeleton for Pairwise Biological Sequence Alignment," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, pp. 561-570, 2009.

[2] M. N. Isa, K. Benkrid, T. Clayton, C. Ling, and A. T. Erdogan, "An FPGA-based parameterised and scalable optimal solutions for pairwise biological sequence analysis," presented at 2011 NASA/ESA Conference on Adaptive Hardware and Systems (AHS), 2011.

[3] T. F. Oliver, B. Schmidt, and D. L. Maskell, "Reconfigurable architectures for bio-sequence database scanning on FPGAs," *IEEE Transactions on Circuits and Systems II*, vol. 52, pp. 851-855, 2005.

[4] Y. Yoshiki, M. Yosuke, M. Tsutomu, and K. Akihiko, "High Speed Homology Search Using Run-Time Reconfiguration," in *Proceedings of the Reconfigurable Computing Is Going Mainstream, 12th International Conference on Field-Programmable Logic and Applications*: Springer-Verlag, 2002.

[5] Y. Yamaguchi, et al., "FPGA-Based Smith-Waterman Algorithm: Analysis and Novel Design Reconfigurable Computing: Architectures, Tools and Applications," vol. 6578, *Lecture Notes in Computer Science*: Springer Berlin / Heidelberg, 2011, pp. 181-192.

[6] K. Dohi, K. Benkrid, C. Ling, T. Hamada, and Y. Shibata, "Highly efficient mapping of the Smith-Waterman algorithm on CUDA-compatible GPUs," presented at 21st IEEE International Conference on Application-specific Systems Architectures and Processors (ASAP), 2010.

[7] Y. Liu, D. L. Maskell, and B. Schmidt, "CUDASW++: optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units," *BMC Res Notes*, vol. 2, pp. 73, 2009.

[8] Y. Liu, B. Schmidt, and D. L. Maskell, "CUDASW++2.0: enhanced Smith-Waterman protein database search on CUDA_enabled GPUs based on SIMT and virtualized SIMD abstractions," *BMC Res Notes*, vol. 3, pp. 93.

[9] Y. Munekawa, F. Ino, and K. Hagihara, "Design and implementation of the Smith-Waterman algorithm on the CUDA-compatible GPU," presented at BioInformatics and BioEngineering, 2008. BIBE 2008. 8th IEEE International Conference on, 2008.

[10] R. Durbin, Eddy, S., Krogh, A., Mitchison, G, *Biological Sequence Analysis: Probabilistic Models for Proteins and Nucleic Acids*: Cambridge University Press, Cambridge UK, 1998.

[11] S. Henikoff and J. G. Henikoff, "Amino acid substitution matrices from protein blocks," *Proceedings of the National Academy of Sciences*, vol. 89, pp. 10915-10919, 1992.

[12] M. O. Dayhoff, R. M. Schwartz, and B. C. Orcutt, "{A model of evolutionary change in proteins}," *Atlas of protein sequence and structure*, vol. 5, pp. 345-351, 1978.

[13] "Virtex-5 Family User Guide," Xilinx, Inc., San Jose, CA 2009.

High Performance Gapped BLAST with the Two-hit Method Implementation on FPGA

M.Nazrin M.Isa, Khaled Benkrid, Thomas Clayton
System Level Integration Group,
School of Engineering,
University of Edinburgh,
EH9 3JL, Edinburgh
{m.n.isa,k.benkrid,t.clayton}@ed.ac.uk

ABSTRACT

A fully-pipelined hardware implementation of the BLAST algorithm is presented. Unlike other reported BLAST with the two-hit method architectures, the proposed core uses fixed configuration elements (CEs) i.e. only two in the PE for alignment matrix computation in a folded PE systolic array. An efficient scheduling strategy based on the double buffering technique is used to alternately hold substitution matrix scores for alignment matrix computation in the CE for the case of computing alignment in multiple-pass. The CE is made up from the abundant logic slices in FPGA instead of block RAM to enable hardware realization without depending on the restricted resources. Implementation results show that the fully-pipelined BLAST architecture with the efficient scheduling strategy successfully achieved a tenfold average speed-up against the NCBI BLAST 2.2.27 with fold factors up to 20 folds and normalized speed-up up to 11x against state-of-the-art hardware implementation.

Keywords: Field Programmable Gate Arrays, Sequence Alignment, Folded Systolic Array, Substitution Matrix, Smith Waterman, Needleman-Wunsch, BLAST, Double Buffering.

1. INTRODUCTION

Searching for an unknown sequence (query sequence) against huge biological databases is repetitive and time-consuming task in molecular biology. With the increasing sizes of genomic datasets in the database, sub-optimal alignment algorithms include the *Basic Local Alignment Search Tool* (BLAST) increasingly gaining popularity over the optimal solutions such as the Smith-Waterman [1] and the Needleman-Wunsch [2] algorithms. BLAST was introduced by Altschul et al. in 1990 [3]. Its search sensitivity and speed was further improved in 1997. This newer version of BLAST is known as the BLAST with the two-hit method [4]. In general, BLAST comprises of three main stages; the seed generation, ungapped extension and gapped extension.

1.1 Seed Generation

In this stage, a query sequence is pre-processed to develop overlapping sub-residues known as *W*-mers, where *W* is the fixed length of the sub-residues. In the case of protein sequence alignment, *W*=3 for BLASTp algorithm while for DNA sequence alignment, *W*=11 in the BLASTn algorithm. Figure 1 illustrates the pre-processing methodology for the case of BLASTp algorithm.

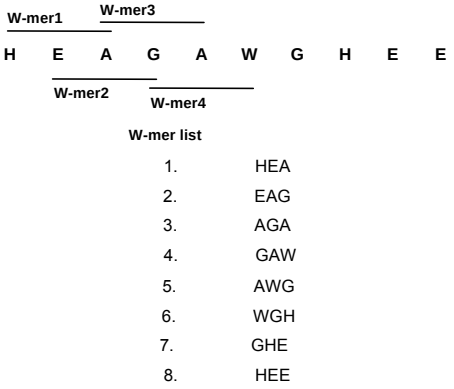


Figure 1. Pre-processing of a query sequence *HEAGAWGHEE* into a list of *W-mer, W=3.*

The number of *W-mers* in the list can be determined by *(q-W)+1* where *q* is length of the query sequence. In this example, *q* = 10 and *W*=3. Then the total number *W-mers* generated is 8. In this stage the preprocessed words are scored against subject sequences in the database using a scoring matrix such as BLOSUM 62. Any word with accumulated score satisfying a given threshold, *T* (typically 11) is recorded for subsequent stage. These word pairs are referred to as hits or highly similar residues of size *W* between the query sequence and subject sequence.

1.2 Ungapped Alignment

In this stage, any non-overlapping hits from the previous stage which have distance *A* (typically 40) within one another that lies on the same diagonal is selected for the ungapped extension. Any two-hit pair satisfying these conditions is then extended in both directions as illustrated in Figure 2. The extension occurs without allowing any gap by scoring the hit pair with the subject sequence. In Figure 2, the solid rectangles represent a pair of the two-hit while the dashed rectangle marks the extension. The extension starts by first closing the gap between the two-hit from right to left as shown by arrow 1. Then the extension from the center of the alignment proceeds to both directions as indicated by the left and right arrows respectively, which proceeds outwards of the corresponding hit position. Similarly as previous stage in section 1.1, a score matrix is used to reward score for any match/mismatch of a particular subject sequence and query sequence residue pair. The score beginning from the start

of extension is accumulated until it drops more than *X* below the maximum-score-so-far or sometimes referred to as *X*-drop mechanism. Any two-hit pair with score exceeds another predefined threshold value is called as high-scoring pair or HSP, which is reported as significant hit for the gapped alignment.

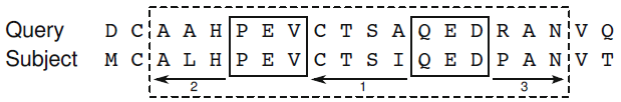


Figure 2. Example of an ungapped extension of a two-hit in the NCBI BLAST Implementation [5]

1.3 Gapped Alignment

The gapped extension operation performs a modified version of dynamic programming (DP) algorithms such as the Needleman-Wunsch and the Smith-Waterman. Details of these algorithms are discussed extensively in [1]. Figure 3 illustrates the gapped alignment of two biological sequences; *broad bean leghemoglobin I* and *horse β-globin* using the BLAST with the two-hit method.

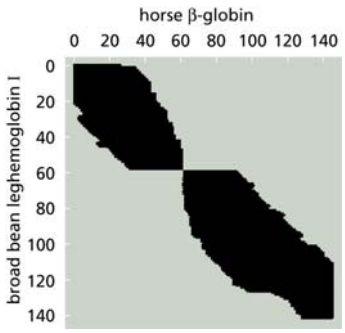


Figure 3. Example of gapped extension of *broad bean leghemoglobin I* and *horse β-globin* [6]

The extension starts from the central point of the HSP generated by the inward ungapped extension and proceeds towards both ends. The *X*-drop mechanism creates a region of the path toward the left and the right apart from the HSP as illustrated in Figure 3. This reduces the running time needed to compute the gapped extension.

2. RELATED WORK

Earlier work of BLAST acceleration used high performance multiprocessor clusters such as the BlueGene [7] and the IBM Blade Cluster [8]. Although the algorithm was scalable to the number of clusters to achieve substantial speed-up performance, the maintenance, energy, costs and size of the accelerator were comparable with single-node solutions [9]. Over a couple of decades ago, FPGA has been extensively used in sequence alignments. The reported work in [10] was among the earlier works of BLAST acceleration in FPGA. This Mercury BLASTn searched DNA sequences using two different FPGAs. A more sophisticated implementation, which searched for protein sequences using the heuristics approach using FPGA was presented in [9]. In this hardware architecture, both the seed generation and ungapped extension of the BLASTp stages were implemented on the XC2V6000 FPGA, while the gapped alignment was implemented in the host CPU. The first two stages ran at 110MHz and 85MHz respectively. Due to limited number of block RAMs of the Virtex-II FPGA, the two stages were implemented onto two FPGAs. Among the three stages of BLAST, the first stage accounts 50 percent of its execution time [9]. Therefore, in order to achieve higher speed-up performance, the other two stages need for acceleration. The studies in [11] and [12] presented their efforts to incorporate the three stages in hardware to gain higher computation performance. However, the query pre-processing stage to generate the hash table was implemented in the host and empirical number of hit finder units was used due to limited hardware resources. On the other hand, Jacob et al. in [12] implemented the hash table in the Xilinx Virtex-II 6000 FPGA by storing it in the off-chip memory and only the pre-filtering stage of the gapped extender was implemented in hardware. To best of our knowledge, none of the reported FPGA implementations parallelized all the three stages due many factors include the restricted amount of block RAM to store a substitution matrix scores inside the PE pipeline [12], [13]. In this work, a fully-pipelined BLAST architecture with the new query pre-processing strategy and the efficient scheduling strategy of the fixed CEs are proposed. Section 3 first details the scheduling strategy followed by presentation on hardware realization of the BLAST stages in section 4. Section 5 discusses the core's performance before conclusions and future work are laid out.

3. THE EFFICIENT SCHEDULING STRATEGY

The proposed hardware architecture utilizes only two CEs in the PE. In order to optimize logic resources in the folded systolic array architecture, these two configuration elements are used alternately for alignment matrix computation in multiple-pass computation. This enables efficient use of logic resources instead of replicating the configuration elements in the PE. Therefore, to effectively manage the fixed CE, the double buffering technique is adopted in the proposed architecture, whereby as the first CE being use to hold coefficients for alignment matrix computation and the other CE is simultaneously configured with new coefficients for subsequent pass computation. This way, subsequent alignment computation continues seamlessly without additional overhead time to configure the CE. In this work, this scheduling strategy is referred to as overlapped computation and configuration (OCC). Details of the OCC strategy are illustrated in Figure 4. To allow for efficient scheduling, initially all CE_0 elements in the pipeline is configured with coefficients during the *Initial Config.* phase. This is the only non-overlapping configuration operation. Once the first pass (F_1) computation starts, CE_1 in the pipeline is updated with new coefficients for subsequent fold computation (labeled as *Overlap 1*). This overlapping operation continues until

all subject sequences in the database are exhausted. Note that, during *Overlap 4*, CE_0 is updated with new coefficients for the next subject sequence.

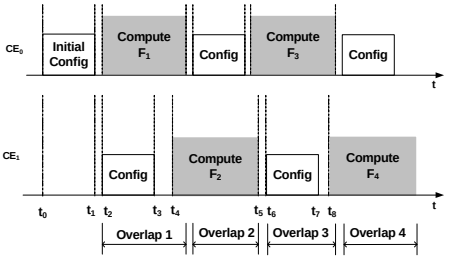


Figure 4. Efficient scheduling strategy between PE computation and CE configuration

4. HARDWARE ARCHITECTURE

The overall hardware architecture of the BLAST algorithm is illustrated in Figure 5. The *Hit Finder*, *Ungapped Extender* and *Gapped Extender* pipeline consist of n_{PEs} where *n* is parameterizable to any number of PEs depending on FPGA device in hand.

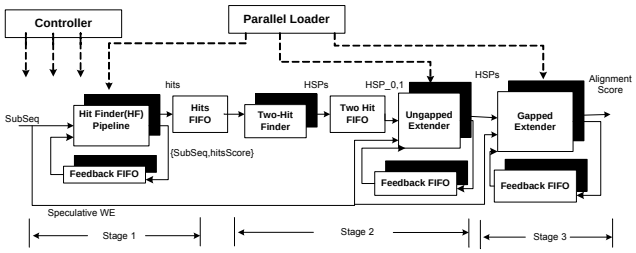


Figure 5. Illustration of BLASTp hardware pipeline

In the first stage, hits are calculated as subject sequence residues are shifted systolically through the PEs at every clock cycle. For the case of protein sequence alignment, *W* is set to 3 residues, whereby for each overlapping three words, score is calculated and those satisfying a given threshold are stored into the *Hits FIFO*. The *Two-Hit Finder* block comprises of *p Two-Hit Finder* units. These units search for significant hits for the ungapped alignment step. Addresses (query and subject sequence residues) of the filtered hits are then stored in the *Two-Hit FIFO*. Since all stages require score matrix in the PE to calculate scores, the *Parallel Loader* as shown in Figure 5 is designed in each stage to simultaneously update all CEs with the worst case configuration of *2xCEDepth* clock cycles. Computation of each stage occurs as subject sequences flow through the pipeline. Details of each block are discussed in the following sections.

4.1 Hit Finder

Figure 6 illustrates the internal architecture of the PE *Hit Finder* unit, where each PE able to hold two query residues in a folded systolic array. For the case of computing alignment matrix in multiple-pass processing, the configuration element (CE) in the PE is used alternately for hit calculation following the scheduling procedure as outlined in section 3.

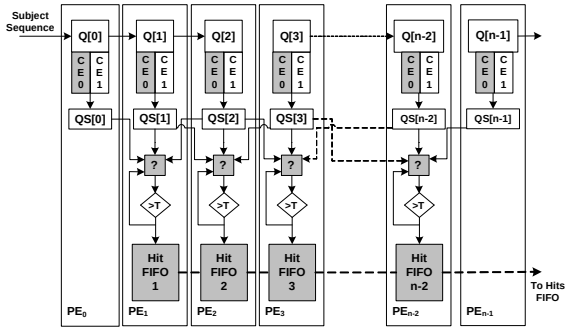


Figure 6. The hit finder pipeline

Each FIFO in the PE stores addresses of the query and subject sequence residues whenever their hits scores exceeds a given threshold value. For the case of BLASTp, the hits score is set to 11. The hit finder is designed to be parameterizable, so that, any threshold value can be used during the hit finding operation. The size of the FIFO is also parameterizable to enable user to change the word length of the query and subject sequence addresses depending on block RAM resources available. As the hit scores of the current subject sequence calculated, the hit FIFO located in each PE shifts their corresponding hits addresses through a serial chain as shown by the horizontal dotted line. All of these addresses are stored in the main *Hits FIFO*.

4.2 The Two Hit Finder

The hit finder unit is essentially a diagonal line checker, which is used to select only meaningful hit pairs satisfying conditions as outlined in section 1.2. This is done by fetching the accumulated addresses in the *Hits FIFO* unit into the *Two-Hit Finder* in parallel. Note that, as the hit finder finishes computing hit scores, new subject sequence is fetched for the next hit finding process or in the case of multiple-pass computation, hit scores for subsequent fold is computed. In this architecture, the numbers of two-hit finder unit is parameterizable and each of them runs in parallel to simultaneously search for meaningful hits from the recorded hits. This approach speeds up the search for HSPs in the ungapped alignment stage. Figure 7 illustrates the functional diagram of a two-hit finder unit.

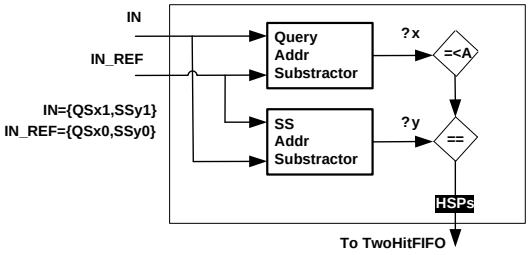


Figure 7. The two hit finder

Two substractors are used; one determines net distance (Δx) between the x-direction i.e. the query address whilst the other calculates Δy for the y-direction i.e. the subject sequence address. If Δx exceeds a given threshold value, *A*

and if Δx equals to Δy, the pair is said to be on the same diagonal line. Then, the address (query and subject sequence addresses) of the HSP is store into the Two-Hit FIFO for subsequent stages, i.e. ungapped extender.

4.3 Ungapped Extender

The ungapped extender block implements the ungapped extension operation as discussed in section 1.2. Its internal architecture is illustrated in Figure 8. Each of the recorded HSPs in the *Two-hit FIFO* is read into the ungapped extender in turn.

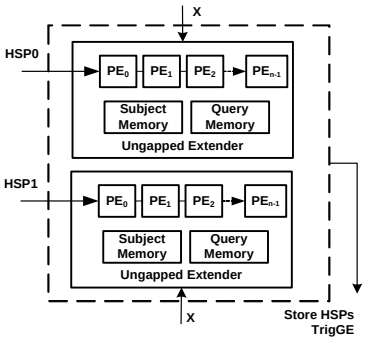


Figure 8. The ungapped extender unit

Since the extension proceeds in two different directions, two ungapped extender pipelines are used; where each pipeline has *n* processing elements. The inwards unggaped extension starts following the ungapped extension procedure as discussed in section 1.2. Then, the outward ungapped extension to both directions i.e. left and right takes place (see arrow 2 and 3 in Figure 2). The query-subject residue pairs along the extension are scored using the substitution matrix in the CE. The outward extension to both directions occurs in parallel and alignment in either direction terminates if any of the following conditions occur. 1) The current accumulated score falls below the cut-off value *X* compared to the maximum-score-so-far. 2) The extension reaches end of the query or current subject sequence.

4.4 Gapped Extender

The hardware architecture for the gapped extender unit is similar as the ungapped extender except in two respects; 1) the gapped alignment starts from the central pair of the inward alignment and the extension proceeds outwards from both left and right directions. 2) The internal PE architecture is slightly more complicated than the one implemented for the hit finder and ungapped extender units. This is because PE with the gapped extension implements elementary calculation of the dynamic programming algorithms for the gapped alignment as discussed in section 1.3.

5. PERFORMANCE EVALUATION

The BLASTp core architecture has been captured using Verilog HDL in a parameterizable manner. The designed core is implemented on the Alpha Data board with Xilinx Virtex-5 FPGA on it. The XC5VLX110-3FF1153 device offers a total of 17, 280 logic slices or 110,592 logic cells and 256 blocks RAM of size 18Kbit each. In terms of throughput performance, speed-up is used to evaluate the performance of the proposed core against the corresponding software implementation, i.e. the NCBI BLASTp. The speed-up is calculated by dividing the execution time of the NCBI BLASTp (t_{BLASTp}) by the execution time of the proposed core (t_{FPGA}), as expressed in (1)

$$Speedup = \frac{t_{BLASTp}}{t_{FPGA}} \qquad (1)$$

In the case of comparison against the software implementation, a set of different query sequences was extracted from the UniprotKB/Swiss-Prot protein knowledgebase version 2012 [14]. The length of query sequences ranges from 100 up to 2048 residues. The database sequences were also extracted from the UniprotKB/Swiss-Prot protein knowledgebase with 538,010 subject sequences or 190,998,508 protein residues. In this evaluation, biological sequences in the database are assumed to be already held in the accelerator card's memory because, in practice, sequence alignment is made against fairly static databases. For fair comparison, the software implementation (i.e. the BLAST 2.2.27+) was executed on the Intel dual core processor (E6600). This desktop computer platform is comparable to the FPGA device used in this comparison as both computation platforms have been fabricated from the 65nm process technology and they come from the midrange type of FPGA and processor respectively. The desktop computer is operated at an operating clock frequency of 2.0 GHz. For the hardware implementation, the designed core was clocked at 200 MHz with 100 PE Hit Finders and 50 PEs for the ungapped extender and gapped extender respectively. The corresponding implementation results of both platforms are summarized in Table 1. Based on the experimental work, the designed core with various folding factors (i.e. up to 20), achieved tenfold average speed-up against the NCBI BLASTp. The multiple-pass processing (with fold factors as in Table 1) is required in this case, due to the limitations of current hardware resources.

Table 1: Speed-up performance of the proposed core against BLAST 2.2.27+

Query Accession # (length)	PEs	# Fold	H/W (s)	Software (s)	Speed Up
P02652(100)	100	1	0.18	6.37	11.91
C5DTC6(222)	111	2	0.75	10.73	14.31
P00762(246)	82	3	1.24	12.82	10.33
P0C9N5(376)	94	4	1.97	20.39	10.35
Q9LU36(570)	95	6	2.95	69.52	23.61
B3KY11(800)	100	8	3.92	56.34	14.37
A8KA62(1000)	100	10	4.93	103.62	21.04
D3DNT2(1600)	100	16	7.71	95.33	12.37
Q9BYP7(1800)	100	18	8.98	121.29	13.51
Q8IYD8(2048)	103	20	9.81	203.93	20.79

For comparison against other FPGA implementations, previous studies in [5], [9], [11] and [15] are used as reference. Due to the different datasets reported in literature, straight comparison against other FPGA implementations cannot be made. For instance, in [5] BLASTp was implemented on the Spartan XC3S500 FPGA with concurrent processing of several queries. Other factors include the use of a group of query sequences as in [5], [9], [15] and [16]. To fairly evaluate other BLASTp implementations in hardware with the designed core, each of the reported cores would have to be implemented on the same FPGA device and tested using the same sets of query and database sequences. However, this is not possible due to having no access to the cores used in the reported FPGA implementations. Alternatively, the best reported runtimes presented in the literature are calculated taking into account different sizes of databases and lengths of query sequences used. Table 2 summarizes the core performance for single pass computation against other FPGA implementations. It is obvious that the proposed core achieves at least 3x speed-up performance against other FPGA implementations.

Table 2: Speed-up performance of the proposed core against other BLASTp implementation on various FPGAs. Selected query length of 100 residues, DB 538,010 sequence, 190,998,508 residues.

Reference	Year	Device	Freq (MHz)	T_{exe} (s)	Speed-Up
[5]	2012	XC3S5000	100	3.70	20.56
[9]	2007	XC2V6000	15	0.64	3.56
[16]	2006	XC4VLX160	100	2.15	11.94
[15]	2007	XC4VFX140	100	0.79	4.39
[11]	2008	XC4VLX160	20	7.89	43.83
Proposed	2012	XC5VLX110	200	0.18	1.00

In an effort to measure speed-up performance of the designed core independently of area and fabrication process technology used, the speed-up figures in Table 2 are normalized with respect to area and the FPGA's look-up table (LUT) delay. In terms of area utilization, the studies cited in Table 2 reported their PE utilization in the form of logic slices, whereas the internal CLB slice architecture varies depending on types and families of FPGAs. For instance, one Virtex-5 FPGA slice is equivalent to two slices of its predecessors beginning from Virtex-4. Each Virtex-5 slice has four logic cells (LCs), whereas earlier generations of Virtex FPGAs have only two LCs per slice [17]. Among other elements, an LC is made up of a LUT, a register and a multiplexer. The logic cell is an abstract logic resource that measures area utilization independently of slice architecture in any particular FPGA family. Therefore, LC is used rather than number of slices as an area normalization factor in order to effectively evaluate speed-up performance across different types of FPGAs. Then, the total LCs used in the PE is calculated using the aforementioned LC-slice relationship.

In addition, the amount of block RAM used in the PE to store substitution matrix scores also has to be taken into consideration prior to the normalization. To take into account memory utilization in the form of LCs, both the Gapped Extender PE and the Feedback FIFO of the designed core are synthesized using Cadence Build Gates version 2005 with 0.18um UMC process technology and the gate counts of each unit is noted. Firstly, the equivalent gate counts-LCs relationship of the Gapped Extender PE is calculated. The total gate counts utilized by the PE is extracted from the Cadence Build Gates tool, while total utilization in terms of logic cells is determined by multiplying the reported slice utilization in the Xilinx ISE 13.1 place and route report by two (for Virtex-4 or older FPGAs) or four (for Virtex-5 and newer FPGAs). Based on these relationships (gate counts and LCs), it is noted that one LC is equivalent to 443 gates. Then, using the gate counts-LCs relationship, equivalent LCs/Kbit of the Feedback FIFO (memory utilization) is calculated by dividing the total number of gate counts of the Feedback FIFO by 443 gates. From this analysis, it is noted that, one Kbit of memory (block RAM) utilizes an equivalent of 8174 gates or 18 LCs. Ultimately, the established LC relationships between both the logic and the memory-based elements are used as a reference benchmark to effectively normalize speed-up performance taking into account both the logic and memory elements used in the form of logic cells. The normalized speed-up performance of the respective FPGA implementations is summarized in Table **3**. The area ratio is calculated by dividing the total LCs of the proposed core with the total LCs of each of the reported FPGA implementations. The core's performance per LC is then calculated by dividing the raw speed-up in Table 2 by the area ratio. In this analysis, the designed core achieved at least 7x speed-up normalized per area compared to others. To evaluate the core's performance independent of fabrication technology, the ratio of the LUT delay of the corresponding FPGA implementations is calculated.

Table 3: Normalized speed-up performance per area and process technology of the proposed core against other FPGA implementations

Ref.	Total Area (#Logic Cells)	Area Ratio	LUT Delay Ratio	Speed-up/ Area	Speed-up/Area/ Process technology
[5]	62,786	1.70	0.19	12.09	2.30
[9]	143,700	0.48	0.23	7.42	1.71
[16]	200,724	0.53	0.53	22.53	11.94
[15]	305,184	0.35	0.53	12.54	6.65
Proposed	106,668	1.00	1.00	1.00	1.00

Prior to that, the LUT delay ratio of the respective FPGA devices used in this comparison is calculated by dividing the LUT delay of the Virtex-5 FPGA with LUT delay of each device. Then, the area normalized speed-up is multiplied by the LUT delay ratio to get the normalized speed-up per area per process technology. Based on the normalized figures, the overall speed-up of the BLASTp core with fixed CEs achieved at least 1.7x against others. The normalized figure shows that the designed core outperformed others independent of area and process technology.

6. CONCLUSIONS

The gapped BLASTp with the two-hit method core architecture has been designed using Verilog HDL with all the BLAST stages are pipelined together to achieve higher performance. A fixed number of configuration elements (only two) have been designed from FPGA's logic slices to optimize logic resources required in the folded systolic array architecture. This has successfully addressed the block RAM limitation as reported in FPGA-based BLASTp implementations. An efficient scheduling strategy based on the double buffering technique is implemented in the core architecture to effectively manage the fixed CEs in a folded systolic array, where the CEs are used alternately in a multiple-pass processing. Implementation results showed that the designed core achieved tenfold average speed-up as compared to the BLAST 2.2.27+ 'software only' implementation which ran on a comparable desktop computer. In the case of comparison with other reported FPGA implementations, the normalized performance indicator (speed-up/logic cells/process technology) was proposed to effectively compare the designed core against others. The results showed that, the proposed core with the efficient scheduling strategy achieved up to 11x speed-up. Since the design is not constrained by any particular FPGA device, and also supports a scalable number of PE systolic arrays, the designed core can be redeployed onto other denser FPGAs for higher performance with minimal design effort. This includes the XC6VLX760 FPGA, which offers 7x higher logic density.

7. REFERENCES

[1] R. Durbin, Eddy, S., Krogh, A., Mitchison, G, *Biological Sequence Analysis: Probabilistic Models for Proteins and Nucleic Acids*: Cambridge University Press, Cambridge UK, 1998.

[2] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of Molecular Biology,* vol. 48, pp. 443-453, 1970.

[3] S. F. Altschul*, et al.*, "Basic local alignment search tool," *Journal of Molecular Biology,* vol. 215, pp. 403-410, 1990.

[4] S. F. Altschul*, et al.*, "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs," *Nucleic Acids Res. 1997 Sep 1;25(17):3389-402.,* 1997.

[5] L. Bleris*, et al.*, "Improvement of BLASTp on the FPGA-Based High-Performance Computer RIVYERA," in *Bioinformatics Research and Applications*. vol. 7292, ed: Springer Berlin Heidelberg, 2012, pp. 275-286.

[6] S. F. Altschul*, et al.*, "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs," *Nucleic Acids Res,* vol. 25, pp. 3389 - 3402, 1997.

[7] E. L. Huzefa Rangwala, Roy Musselman, Kurt Pinnow, Brian Smith, Brian Wallenfelt, "Massively parallel BLAST for the Blue Gene/L," in *High Availability and Performance Computing Workshop*, 2005.

[8] L. Heshan*, et al.*, "Efficient Data Access for Parallel BLAST," in *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, 2005, pp. 72b-72b.

[9] A. Jacob*, et al.*, "FPGA-accelerated seed generation in Mercury BLASTP," in *Field-Programmable Custom Computing Machines, 2007. FCCM 2007. 15th Annual IEEE Symposium on*, 2007, pp. 95-106.

[10] P. Krishnamurthy*, et al.*, "Biosequence similarity search on the Mercury system," in *Application-Specific Systems, Architectures and Processors, 2004. Proceedings. 15th IEEE International Conference on*, 2004, pp. 365-375.

[11] S. Kasap*, et al.*, "High performance FPGA-based core for BLAST sequence alignment with the two-hit method," in *8th IEEE International Conference on BioInformatics and BioEngineering, 2008. BIBE 2008.* , 2008, pp. 1-7.

[12] A. Jacob*, et al.*, "FPGA-accelerated seed generation in Mercury BLASTP," in *15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2007. FCCM 2007.* , 2007, pp. 95-106.

[13] M. Atabak and C. H. Martin, "Fast and accurate NCBI BLASTP: acceleration with multiphase FPGA-based prefiltering," presented at the Proceedings of the 24th ACM International Conference on Supercomputing, Tsukuba, Ibaraki, Japan, 2010.

[14] UniProtKB/Swiss-Prot. (2012). *UniProtKB/Swiss-Prot protein knowledgebase release 2012_09*. Available: http://web.expasy.org/docs/relnotes/relstat.html

[15] E. Sotiriades and A. Dollas, "A General Reconfigurable Architecture for the BLAST Algorithm," *The Journal of VLSI Signal Processing,* vol. 48, pp. 189-208, 2007.

[16] J. C. Herbordt*, et al.*, "Single Pass, BLAST-Like, Approximate String Matching on FPGAs," in *14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2006. FCCM '06.* , 2006, pp. 217-226.

[17] Xilinx, "Xilinx UG190 Virtex-5 FPGA User Guide," 2007.

A Run-time Reconfigurable System for Adaptive High Performance Efficient Computing

Chuan Hong, Khaled Benkrid, Nazrin Isa and Xabier Iturbe
System Level Integration Group, School of Engineering, The University of Edinburgh, UK
{C.Hong, K.Benkrid, M.N.Isa, X.Iturbe}@ed.ac.uk

***Abstract*— Field programmable hardware gives electronic systems the ability to be reconfigured at run time. This allows electronic systems to be more efficiently customized on demand and on-the-fly depending on user requirements and environmental changes. This paper presents a run-time reconfigurable system that allows computing tasks to adjust their sizes in response to current available resources, optimizing the overall performance by maximally exploiting all the resources on the chip. In particular, we present a novel run-time task assembler, which assembles tasks with desired parameters on-the-fly, together with an efficacious run-time task placer to rapidly configure tasks at optimum locations. The system is demonstrated with a dynamic programming-based pairwise sequence alignment application. Real hardware implementation result shows that our run-time reconfigurable system optimizes resource usage on the fly by ~ 3x, while matching the performance of carefully hand-crafted static solutions.**

***Index Terms*— Reconfigurable Computing, Adaptive Hardware, High Performance Computing, Bioinformatics**

I. Introduction

FIELD programmable hardware gives electronic systems the ability to change their function and configuration after manufacture. By enhancing configurability, the reconfiguration overhead can be reduced in terms of both speed and area, which makes rapid run-time reconfiguration possible [1]. This allows electronic systems to be flexibly reconfigured with desired parameters on-the-fly, therefore hardware resources are more efficiently utilized, by trading off performance with hardware resources and power consumption [2].

However, due to the constraints imposed by increasingly heterogeneous hardware, most existing approaches lack flexibility in terms of the size of reconfigurable tasks and the size of Partial Reconfiguration (PR) regions. Most commonly, such approaches use predefined tasks synthesized off-line [3] [4] [5]. As a result, only a limited number of tasks are available. Moreover, memory space is wasted to accommodate multiple versions of bitstreams. In addition, predefined fixed PR regions whereby boundaries have to be statically fixed to accommodate static communication ports (Bus Macros or Proxy Logic) [6] [7], result in wasted resources since PR regions have to cater for the largest reconfigurable task possible.

To circumvent the above shortcomings, we present a run time reconfigurable system, which not only supports on-line task generation, but also allows tasks be placed at arbitrary positions with no boundary concerns. This is achieved by a novel run-time task assembler and a run-time task placer as central components of our reconfigurable system. The task assembler takes into the consideration both user requirements and currently available resources to select hardware resources and assemble them to a full application task on-the-fly; whereas the task placer is responsible for rapidly placing tasks at an optimum position within minimum time overhead by applying state-of-art 2D-packing algorithm with compressed bitstream. The communication between tasks and the host uses a hybrid network, which is based on a high bandwidth bus and an internal configuration port. Our run-time reconfigurable system gives considerably higher flexibility to allow for high performance computing systems to autonomously adapt to user demands and current resource availability. For instance, our system can support scalable multi-user, multi-tasking applications whereby resources can be dynamically managed in respect of user requirements and hardware availability.

We demonstrate our approach in the context of a bioinformatics sequence alignment application [8], in which a query sequence needs to be compared with a database using a pipeline of Processing Elements (PEs). In this context, the number of pipeline stages can be adjusted depending on currently available resources and user requirements. Compared with previous reconfigurable approaches [9] [10], our contribution includes 1) finer grained task scalability and flexible task customization supporting for dynamic multi-user multi-tasking applications; and 2) higher resource usage efficiency and better overall system performance, achieved by adjusting task size with current resource availability.

The remainder of the paper is organized as follows. Section II first outlines the overall proposed run-time reconfigurable system. Then, the run-time task assembler and run-time placer are respectively presented in section III, IV. The sequence alignment application is then demonstrated in section V, with implementation results given in section VI. Finally, conclusions are drawn in section VII.

II. System Architecture

In our proposed system, the whole chip area is divided into two partitions: a static region for the run-time reconfigurable system and a PR region for application tasks. The latter is not constrained by predefined boundaries, which allows hardware tasks to be swapped in/out in a time multiplexed fashion. Fig.1 depicts our proposed system architecture, in which 4 tasks with different pipeline stages are placed and executed in their PR region. The static run-time reconfigurable system consists of a host API, a bitstream

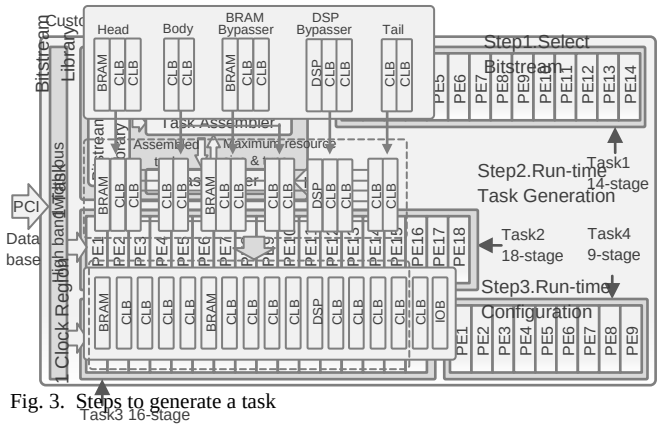


Fig. 3. Steps to generate a task

Fig. 1. Illustration of our proposed run-time reconfigurable system

library, a task assembler and a task placer. The host API is connected in an open network and uses TCP/IP protocols to communicate with clients. After receiving requests from clients, the task information (including task type and required performance) is passed to the run-time task assembler. Depending on the user requirement, the task assembler first decides the type of basic processing elements (PEs) and their constructions, e.g. pipeline stages, then reads from the bitstream library to assemble a full application task and pass it to the task placer. If current resources are insufficient for the task, i.e. task size is larger than maximum available free area on the chip, the task assembler will resize the task to fit within the current available resources using a folding mechanism. The task placer analyzes the currently available chip resources to attempt an optimum position for the task. The optimum position is calculated using state-of-art 2D packing algorithms, which gives more compact placements with less fragmentation. To allocate tasks, the placer uses the Internal Configuration Access Port (ICAP) with compressed configuration mode to reduce the configuration time overhead. After a task is placed on the chip, the placer will update the information of current available resource (maximum available resource size and type), which is then feedback to task assembler to be used for assembling next upcoming task. The communication between tasks and the host uses a hybrid network, which consists of a high bandwidth bus (high throughput) and the ICAP (low throughput). Tasks requiring high throughput communication are connected to the bus, whereas tasks with low external interaction can be placed anywhere and use ICAP for input/output data. The ICAP based communication utilizes the configuration port to write/read data to/from tasks through the configuration layer, which maintains a routing-less PR region [11]. In our real hardware implementation, the host API is implemented on a Xilinx MicroBlaze microprocessor; whereas the task assembler and task placer are implemented separately on two Xilinx PicoBlaze processors.

III. RUN-TIME TASK ASSEMBLER

The run-time task assembler assembles pre-synthesized PEs to generate functional application tasks. To enable the inter-communication between separately synthesized PEs, an inter-PE Bus Macro (BM) is integrated before synthesis. This is a module that has been manually placed and routed using particular wires. The BM constrains the PEs to use specified wires for both inputs and outputs. Therefore, the PEs are able to communicate with each other as long as the same wires are shared between the previous PE's output and the next PE's input within the pipeline. Fig.2 gives an example of the inter-PE communication in the case of a Xilinx Virtex-5 FPGA. The FPGA resources are firstly divided by vertically aligned clock regions, and each clock region is divided by horizontally aligned columns, including Configurable Logic Block (CLB) columns and Block RAM (BRAM) columns [12]. The short wires are homogeneously distributed between two adjacent CLBs. In this example, PEs are area-constrained within two CLB columns. The horizontal short wires (direct lines) are shared between PE1 and PE2, which exclusively connect two CLBs. To direct the output signal from a flip-flop or a Look-Up-Table (LUT) to the specific wires (direct lines), the output signal hops among Programmable Interconnections Points (PIPs) within a switch box before terminating at direct lines. Since the switch boxes routings are highly regular in each column despite resource heterogeneity, the direct lines can be used between any two columns, which give a constant propagation delay. Xilinx Virtex5 FPGAs have 6 direct lines between each two adjacent CLBs, which provide sufficient bandwidth for regular tasks.

In our proposed run-time system, a task is generally composed by a task header, task body, task tail and dummy columns. The task header is usually implemented in a BRAM column which contains the input data for the task. The task body consists of pipelined PEs, whose length is determined by user requirement and current available resources. The task assembler will find the maximum length of PEs to meet both user requirement and current resource availability, otherwise the task will wait in the task queue until previously placed task is finished and removed from the chip. The task tail

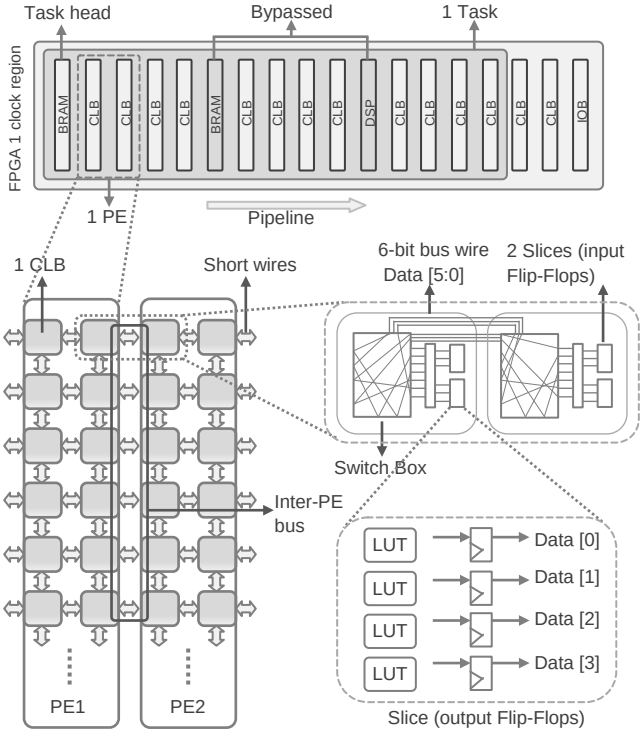


Fig. 2. Inter-PE communication illustration

outputs processed data, which could be implemented using BRAM or CLB depending on the data size. According to the resource type information received from task placer, the

dummy columns are used to cope with heterogeneities within the FPGA chip, where for instance CLB columns in Virtex-5 FPAs are intercepted by other types of columns e.g. BRAM, DSP and IOB. To pass data from a previous PE to the next PE without any data processing, the dummy columns (bypasser) are separately synthesized to directly connect their inputs to their outputs using switch boxes only. Fig.3 gives the steps to generate a customized application task by dynamically assembling PEs from the bitstream library.

IV. Run-time Task Placer

Since assembled tasks are expected to have a high variation in their sizes, chip resources could be severely fragmented if tasks are randomly placed. To floor-plan the chip resources and place tasks at an optimum location, a run-time task placer is developed, using state-of-art 2D-packing algorithm and compressed bitstream to reduce the configuration overhead in both area and time, respectively.

A. EAC Placement Algorithm

In our proposed system, the PR is not constrained with communication routings, therefore tasks can be arbitrarily allocated anywhere on the chip. In such context, the FPGA chip resources can be modeled as a 2-dimensional rectilinear grid, where 2D-packing algorithms can be applied. To compute the optimum location within a short period, we have presented Empty Area Compact (EAC) algorithm in [13]. The EAC uses two matrices to represent current chip resource usage. The first matrix is called "Shape Matrix", in which the occupied cells (CLBs) are marked as zero, and all other cells are incrementally scored in the horizontal direction. The second matrix is named "Area Matrix", in which each score is the maximum rectangular size on the top left. A "Row MER" column and a "Chip MER" are attached to show the Maximum Empty Rectangle (MER) at each row and for the whole chip. Fig.4 gives an example of the two matrices, the five grayed cells are the CLBs occupied by previously placed tasks. The size of the dash-circled area is 3×2, with a width of 3 and area of 6, labeled in the bottom-right cell in Shape Matrix and Area Matrix respectively. An upcoming task is compared with both matrices to pre-select all possible locations. After that, each possible location is scored and the position with the minimum cost is determined as the best location to place the task. Simulations show that the EAC algorithm achieves 25% improvement in task acceptance rate compared to previously developed KAMER and Vertex List-based algorithms [13]. After a task is placed, the task placer will update the chip resource information and then feedback the size and resource type of the chip MER to the task assembler for assembling next task.

Fig. 4. Illustration of EAC placer

B. Compressed Bitstream Configuration

Configuring a task onto the chip requires writing bitstream to the configuration port, e.g. the internal Xilinx ICAP port in Virtex-5 FPGAs which is a 32-bit wide, usually clocked at 100MHz [12]. The bitstream is composed by frames, which represent the minimum configuration unit and consist of 41 32-bit words per frame. The frame number required for each type of resource is given in Table I, and Fig.5 depicts the bitstream alignment for one CLB column in Virtex-5. Conventionally, 41-word frames are written sequentially, which consumes 41 cycles for each frame configuration; hence the configuration time is:

$$T_{cfg} = T_{init} + (41 \times N_{frame}) \times f_{clk} \qquad (2)$$

Where T_{cfg} is the whole configuration time, T_{init} is the initialization time, N_{frame} is the number of frames and f_{clk} is the ICAP clock frequency. However, the configuration speed can be significantly improved if the bitstream consists of a number of identical frames. The identical frames can be replicated to another frame address with 2 more cycles (see Fig.6); therefore the time needed to configure identical tasks can be reduced to:

$$T_{cfg} = T_{init} + (41 + 2 \times N_{frame}) \times f_{clk} \qquad (3)$$

The latter is called compressed bitstream configuration, which was originally invented by Xilinx to reduce the bitstream size. In our proposed system, this technique is applied not only for speeding up partial reconfiguration, as pipelines often heavily reuse identical PEs, but also for blanking finished tasks (replicating zero frames), so that power consumption is reduced. When multiple identical configurations are required, the compressed configuration mechanism achieves a maximum of ~20x speed-up compared with conventional sequential configuration.

V. Case Study Implementation: Pairwise Biological Sequence Alignment

A. Pairwise Biological Sequence Alignment

TABLE I
Xilinx FPGA Frame Number

	CLB	BRAM routing	BRAM content	DSP	IOB	CLK
Virtex4	22	20	64	21	30	3
Virtex5	36	30	64	28	54	4

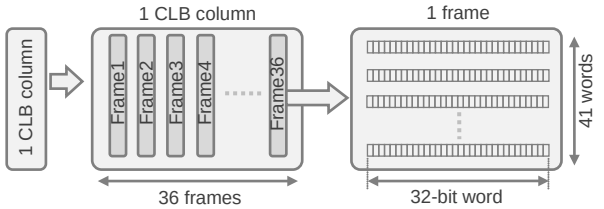


Fig. 5. Bitstream and frame alignment of one CLB column in Virtex-5

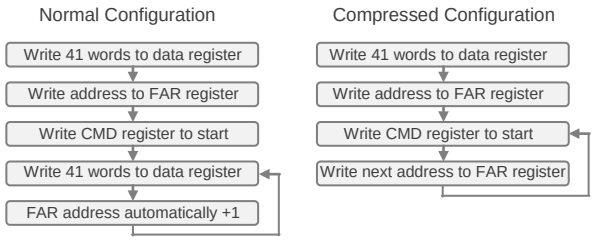


Fig. 6. Steps for normal and compressed configuration

In bioinformatics, sequence alignment is used to identify the regional/global similarities between two biological sequence e.g. DNA, RNA or protein sequences. To achieve the optimum alignment, dynamic programming based sequence alignment algorithms such as the Smith-Waterman algorithm [8] are widely used to find and score the best alignment between a query sequence and database sequences. The affine gap penalty based Smith-Waterman (S-W) algorithm introduced by GOTOH [14] in 1982, is widely used today since it is more biologically realistic than the linear gap penalty model. It uses three matrices calculated as shown below:

$$F(i,j) = \max\begin{cases} F(i-1,j-1)+s(x_i,y_j) \\ I_x(i-1,j-1)+s(x_i,y_j) \\ I_y(i-1,j-1)+s(x_i,y_j) \end{cases}$$

$$I_x(i,j) = \max\begin{cases} F(i,j-1)-d \\ I_x(i,j-1)-e \end{cases} \qquad (4)$$

$$I_y(i,j) = \max\begin{cases} F(i-1,j)-d \\ I_y(i-1,j)-e \end{cases}$$

Where $F(i,j)$ is the best score so far; $I_x(i,j)$, $I_y(i,j)$ are the scores aligned to gaps with residue x_i and y_i respectively; $S(x_i,y_j)$ is the probabilistic score of substituting x_i with y_i or vice-versa; d is the penalty associated with opening a gap, and e is the penalty associated with extending a gap. The probabilistic score for substituting x_i with y_j or vice-versa uses a substitution matrix e.g. BLOSUM50 [8] based on a particular biological model.

B. Hardware Pipeline & PE Folding

The S-W algorithm can be implemented in hardware using pipelined PE arrays for higher performance [15]. The PE array (of length N) is mapped to query residues (one-to-one), and the database sequence (of length M) is shifted into the PE array. After the entire database sequence is shifted through the PE array, the highest alignment scores among the database sequence are retained as they depict the more biologically related sequences to the query sequence. However, limited hardware resources in practice rarely allow for one-to-one mapping between PEs and query residues; hence the available resources need to be reused using folding.

Fig.7 gives an example whereby only three PEs can be fitted in hardware for six-residue query sequence. The three PEs are first assigned with the first three query residues (e.g. 'P' 'A' 'R'). After a database sequence is shifted in, the three PEs are then folded with the next three residues (e.g. 'W' 'D' 'C'). The results from the first fold are buffered into a FIFO to be used as input in the second fold, hence then need for an input multiplexer.

We have designed a reconfigurable PE architecture which allows for the online generation of scalable pipelines with low time and area overheads [16].The architecture of a single PE is presented in Fig.8. Each PE has four inputs ($F(i,j-1)$, $I_x(i,j-1)$, $I_y(i,j-1)$, x_i), four outputs ($F(i,j)$, $I_x(i,j)$, $I_y(i,j)$, x_i), and a 3-clock cycle latency. The substitution values for a particular query residue are stored in one LUT inside the PE, whereas another LUT in the PE is configured with the next folding substitution values.

C. Integration with our Run-Time Reconfigurable System

In our pairwise biological sequence alignment implementation, each PE uses 2 CLB columns (see Fig.9). The top 14 switchboxes are used for inter-PE communications, in which the left-side 7 are used for inputting data from the previous PE and the right-side 7 are used for outputting values to the next PE. Each switchbox has 6 direct lines, which together give a 42-bit wide input, as for the output. At the bottom of each PE, a feedback link (5-bit wide) is integrated to allow the last PE (task tail) to feed its output back to the first BRAM for the next folding.

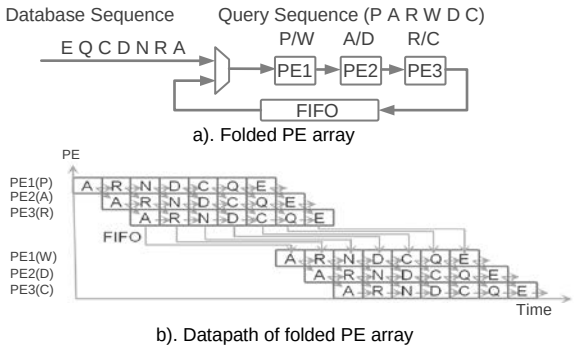


Fig. 7. PE array folding

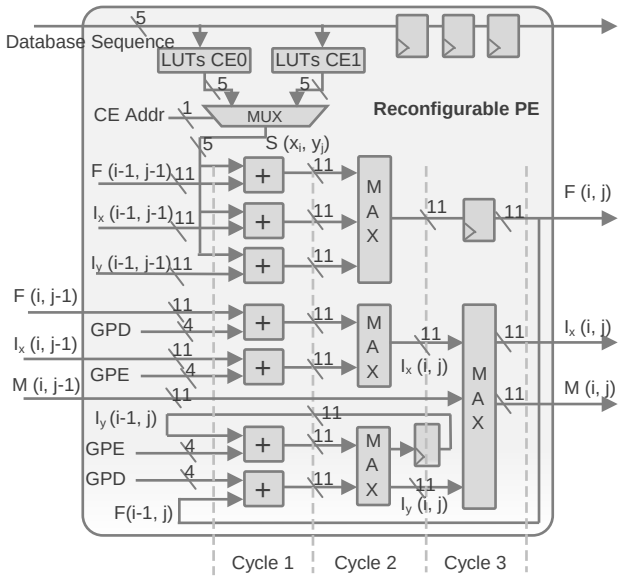


Fig. 8. Reconfigurable PE architecture

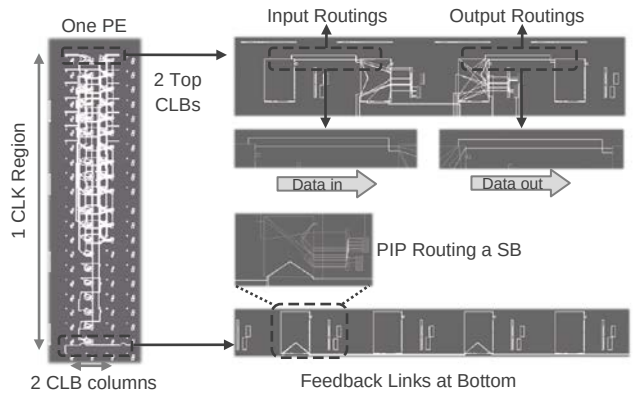


Fig. 9. Implemented PE and communication routing

Our pairwise sequence alignment core was implemented on a Xilinx Virtex5 LX110T FPGA (see Fig.10). The run-time reconfiguration system resides in the static region, leaving all other resources free to be allocated to application tasks. Application tasks can be scheduled and allocated at different positions by the run-time reconfiguration system. For instance, Fig.10 shows two different PE pipeline arrays (21 and 9 PEs in length respectively) running in parallel for two different query sequences (of arbitrary length). In our sequence alignment case study implementation, the PE array starts from a BRAM column (task head), which caches the database sequence from external memory. In the target FPGA chip (Xilinx Virtex5 LX110T FPGA), there are 8×4 BRAM columns, giving 32 possible positions to allocate a task. A high-bandwidth bus is implemented in the leftmost area of the chip for fast update of the first BRAM columns' content. Since alignment scores have relatively small sizes, the ICAP is used for reading task results. The chip has 50×8 CLB columns, which allows a maximum of eight 25-long PE pipelines. To further increase the number of pipeline stages for a single task, a PE array can be routed across clock regions by using a previously developed snake strategy [17], allowing for a 200-long PE pipeline.

VI. Results Analysis

The performance of our hardware implementation was tested on an Alpha Data ADM-XRC-5LX board, which has a Xilinx XC5VLX110T FPGA chip. The performance of PE is tested using UniProtKB/ TrEMBL database and the real results are compared with other software and hardware implementations. Results show that our dynamically online-generated tasks have zero-penalty on the PE performance, compared with carefully hand-crafted static solutions [16]. TABLE II shows that when the maximum pipeline stage (25 PEs) is used, the task achieves a ~30x speed-up, compared with software SSEARCH35 running on Intel(R) Quad Core 64-bit Q8300. TABLE III gives the result compared with other two hardware approaches. In order to compare performance regardless of the technology of the device, such as gate delay and propagation delay, the normalized speed-up is obtained by:

$$SpeedUp_{normalized} = \frac{SpeedUp_{Raw}}{Number_{LC}} \times Device\,Delay$$

Where $SpeedUp_{Raw}$ is the real speed improvement, $Number_{LC}$ is the number of consumed logic cells, which is proportional to the occupied area, and *Device Delay* is the propagation delay of the particular device. Result shows that the normalized speed is improved by 1.53x and 2.4x, compared with other two hardware approaches respectively [10] [18].

Table IV gives the performance of task placer, including the time spent on making location decisions and configuring different size of tasks. The average time for configuring one application task (including placement algorithm execution and hardware reconfiguration times) is less than 150 μs, and a task can be removed within 40 μs. In terms of area, the whole run-time reconfigurable system consumes 1210 slices and 6 BRAMs, which is just 7% and 5% of the total resources on XC5VLX110T FPGA, respectively.

A real-time multi-user, multi-tasking test example is given in TABLE V, in which the overall system run-time performance is improved by adjusting task size (number of folds) with the current available chip resources. In this example, six task sets (φ1- φ6) are used and each task set contains a number of tasks. All the tasks are identically used for P02652 database scanning, but with different user requirements on their performances, namely the number of maximum folds, or the minimum number of PEs. In another word, tasks requiring better performance cannot be folded too many times. During the test, tasks are requested from the user every 36 or 18 seconds, with a different maximum fold number, which is a random number ranging from $Fold_{min}$ to $Fold_{max}$. For example, in set φ1, a task is requested every 36 second from the user, and the number of allowed folds (maximum fold number) for each task is a random number ranging from 4 to 32. Starting from 4 folds (the minimum folds limited by the XC5VLX110T chip size), the folding number is doubled (i.e. task size is halved) every time the available resource is not enough for the current folds. If a task cannot be placed with its maximum fold number, it will wait in the task queue until any previous task be finished and removed from the chip. After one hour, the number of competed tasks ($N_{finished\_tasks}$) is recorded and the completion rate ($R_{finished\_tasks}$) is calculated. Since task size can be adjusted and fit into smaller resource slots, more chip resources are used at the same time, and less resource stays in idle. Benefiting from this, the task finishing rate is increased by ~3x compared with tasks without using folding adjustment (see Fig.11).

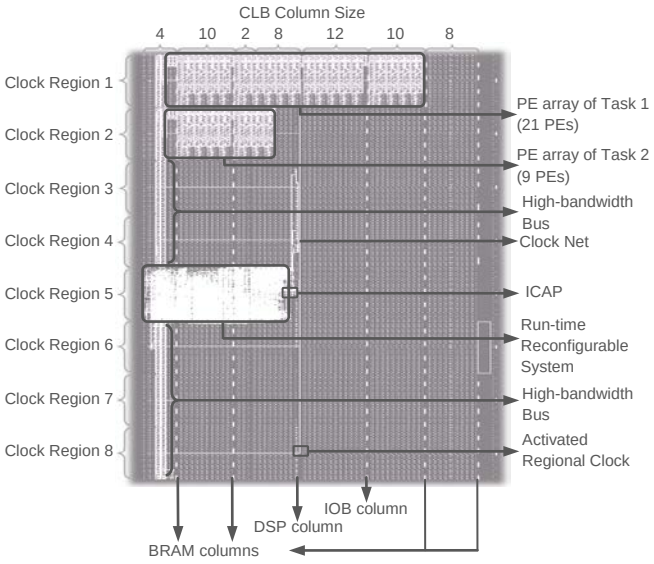


Fig. 10. Implemented Pairwise Sequence Alignment Core on XC5VLX110T with high-bandwidth bus

VII. Conclusion

In this paper, we presented a run-time reconfigurable system which allows application tasks to adapt their sizes depending on current available resources. The system is demonstrated in the context of a sequence alignment application. Results show that tasks can be generated on the fly with minimal time and area overhead, and the overall efficiency is improved by running multiple tasks for multiple users in parallel in a way that is adaptable to currently available hardware resources and user requirements. As for future works, the system adaptability can be extended to give response to other environmental changes, such as heat, power, and emerging faults for low-power and fault-tolerant applications.

TABLE II
PE Execution Time Compared with Software

Query Accession	Length	PE	Fold	Execution Time (s)		Speed-up
				Ours	SSEARCH	
P02652	100	25	4	303	9416	×31.08
Q9H3V2	200	25	8	599	17160	×28.65
Q8NC42	400	25	16	1215	35992	×29.62

TABLE III
PE Speed-Up Compared with Hardware

Ref	Device	Resource Ratio[1]	Device Delay Ratio[2]	Raw Speed-up	Normalized Speed-up
[18]	XC2V6000	0.69	0.23	×4.58	×1.53
[10]	XC2V6000	0.49	0.23	×5.13	×2.40

[1] Resource Ratio = LCs consumed by our approach / LCs consumed by Ref.
[2] Device Delay Ratio = XC5VLX110 delay / XC2V6000 delay

TABLE IV
Task Placer Performance (under 100MHz)

Process	Time
Finding Location (Average Time)	18 µs
Configure PE Header	68.38 µs
Configure 1 PE Body	2.98 µs
Configure whole task (10x PE)	111.18µs
Configure whole task (25x PE)	134.22µs
Update Data Input (Configure one BRAM content)	26.56 µs
Blank whole task (25x PE) (Power Saving)	36.32µs

TABLE V
System Performance Test

Task set	φ_1	φ_2	φ_3	φ_4	φ_5	φ_6
$Fold_{max}$	32	16	8	32	16	8
$Fold_{min}$	4	4	4	4	4	4
$N_{requested\_tasks}$	100	100	100	200	200	200
$R_{requested\_tasks}$	Every 36 s			Every 18 s		
Using folding adjustment						
$N_{finished\_tasks}$	91	73	57	93	78	59
$R_{finished\_tasks}$	91%	73%	57%	46.5%	39%	29.5%
Not using folding adjustment						
$N_{finished\_tasks}$	32	32	32	38	38	38
$R_{finished\_tasks}$	32%	32%	32%	19%	19%	19%

Chip size: 50 × 8 CLB columns, Test duration: 1 hour (3600s)
Query accession: P02652, Folding starting number: 4

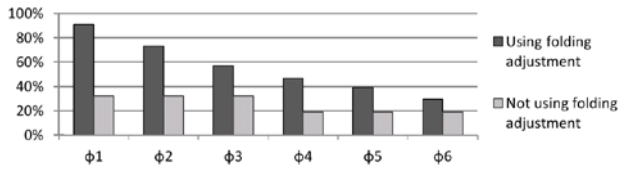


Fig. 11 Performance improvement by using folding adjustment

References

[1] E.L.Horta, J.W.Lockwood, D.E.Taylor, and D.Parlour, "Dynamic hardware plugins in an FPGA with partial run-time reconfiguration," in *Proc. Design Automation Conference*, pp. 343-348, 2002

[2] A. Donato, F. Ferrandi, M. Redaelli, M. D. Santambrogio, and D. Sciuto, "Caronte: a complete methodology for the implementation of partially dynamically self-reconfiguring systems on FPGA platforms," in *IEEE Symposium on FCCM*, pp. 321-322, 2005

[3] X. Iturbe, K. Benkrid, T. Arslan, I. Martinez, M. Azkarate and M. D. Santambrogio, "A Roadmap for Autonomous Fault-Tolerant Systems", in *Proc. Conf. DASIP*, pp. 311-321, 2010.

[4] B.Blodget, S.McMillan, and P.Lysaght, "A lightweight approach for embedded reconfiguration of FPGAs," in *Proc. Design, Automation and Test in Europe Conference and Exhibition*, pp. 399-400, 2003

[5] T. Becker, W. Luk, and Peter Y. K. Cheung, "Enhancing Relocatability of Partial Bitstreams for Run-Time Reconfiguration", *IEEE Symposium on Field-Programmable Custom Computing Machines*, pp. 35-44, 2007.

[6] Xilinx Inc., "Partial Reconfiguration User Guide," *UG702*, October 2010.

[7] P. Sedcole, B. Blodget, J. Anderson, P. Lysaght, T. Becker, "Modular partial reconfiguration in Virtex FPGAs," in *Proc. Int. Conf. on Field-Programmable Logic and Applications*, pp. 211-216, 2005

[8] Durbin R, Eddy, S., Krogh, A., Mitchison, G. "*Biological sequence analysis: probabilistic models of proteins and nucleic acids*" Cambridge University Press, Cambridge UK; 1998.

[9] Y. Yamaguchi, Y. Miyajima, T. Maruyama, A. Konagaya. "High Speed Homology Search Using Run-Time Reconfiguration". *Proc. Field-Programmable Logic and Applications: Reconfigurable Computing Is Going Mainstream*, pp. 671-687, 2002.

[10] T. F. Oliver, B. Schmidt, D. L. Maskell "Reconfigurable architectures for bio-sequence database scanning on FPGAs". *IEEE Trans. Circuits and Systems II*, vol. 52, no.12, pp. 851-855, 2005.

[11] X.Iturbe, K. Benkrid, T. Arslan, R. Torrego, I. Martinez, "Methods and Mechanisms for Hardware Multitasking: Executing and Synchronizing Fully Relocatable Hardware Tasks in Xilinx FPGAs," in *Proc. Int. Conf. on Field-Programmable Logic and Applications*, pp. 295-300, 2011.

[12] Xilinx Inc., "Virtex-5 FPGA Configuration User Guide" *UG191*, 2012.

[13] C.Hong, K.Benkrid, X.Iturbe, A.Ebrahim, and T.Arslan, "Efficient On-Chip Task Scheduler and Allocator for Reconfigurable Operating Systems," *IEEE Embedded Systems Letters*,vol.3, no. 3, pp.85-88, 2011.

[14] O. Gotoh. "An improved algorithm for matching biological sequences". Journal of Molecular Biology, vol. 162, no. 3, pp. 705, 1982

[15] K. Benkrid, A. Akoglu, C. Ling, Y. Song, Y. Liu and X. Tian, "High Performance Biological Pairwise Sequence Alignment: FPGA vs. GPU vs. Cell BE vs. GPP", International Journal of Reconfigurable Computing, April 2012.

[16] M.N.Isa, K.Benkrid and T.Clayton "Efficient Architecture and Scheduling Technique for Pairwise Sequence Alignment" *ACM, SIGARCH Computer Architecture News*,(in press), 2012

[17] X.Iturbe, K.Benkrid, A.Ebrahim, C.Hong, T. Arslan, and I.Martinez, "Snake: An Efficient Strategy for the Reuse of Circuitry and Partial Computation Results in High-Performance Reconfigurable Computing", in *Proc. Int. Conf. on ReConFig*, pp. 182-189, 2011

[18] K. Benkrid, Y. Liu, A. S. Benkrid. "A Highly Parameterized and Efficient FPGA-Based Skeleton for Pairwise Biological Sequence Alignment". IEEE Trans. VLSI Systems, Vol. 17, no. 4, pp. 561—70, 2009

Efficient Architecture and Scheduling Technique for Pairwise Sequence Alignment

M.N.Isa, K.Benkrid and T.Clayton
System Level, Integration Group,
School of Engineering, University of Edinburgh, EH9 3JL, U.K
(e-mail: m.n.isa, k.benkrid, t.clayton@ed.ac.uk).

Abstract— *A novel efficient hardware architecture to optimize the execution time of dynamic programming-based (DP) pairwise sequence alignment algorithms in hardware is proposed. It is realized by introducing an efficient overlapped scheduling of alignment matrix computation and substitution coefficients' pre-loading onto processing elements (PEs) in folded systolic arrays. A new metric is also proposed as an independent performance evaluator to compare different core implementations on different FPGA platforms fairly. Implementation results show that the new hardware architecture for sequence alignment achieves a minimum of 40 percent area normalized speed-up compared to the state-of-the-art hardware implementation, with the speed-up growing linearly with the number of folds e.g. 120 percent speed-up for 16-fold. Compared to equivalent software implementations, the novel hardware architecture achieves a minimum of 103x speed-up, with the speed-up growing linearly with the number of folds e.g. 140x speed-up for 20-fold.*

Index Terms— Algorithms implemented in hardware, Gate arrays, Pipeline processors, Reconfigurable hardware, sequence alignment, bioinformatics.

I. INTRODUCTION

Pairwise Dynamic Programming (DP)-based sequence alignment algorithms such as the Smith-Waterman (S-W) algorithm are widely used to align pairs of sequences as they produce optimal alignments [1]. However, such algorithms have a quadratic complexity both in computation time and memory space. SPLASH [2] was the first off-the-shelf FPGA (Field Programmable Gate Array) used to address these issues. However, during that time, FPGA was not competitive as compared to other computing platforms including a standard desktop computer. Alternatively, a number of parallel architectures have been developed includes SIMD (Single Instruction Multiple Data) architectures such as MGAP[3], Kestrel[4] and Fuzion[5]. This parallel architectures offer a considerably high speed-up performance over a standard desktop solution with expense of design and programming costs. Other reported solutions have used special purpose hardware to accelerate the algorithm by means of processing the DP-based algorithm in parallel using systolic arrays. These include SAMBA (Systolic Accelerator for Molecular Biological Applications) [6], BioSCAN (Biological Sequence Comparative Analysis Node)[7] and BISP (Biological Information Signal Processor)[8]. Due to the non-re-programmability nature of these special purpose architectures, different needs for the implemented algorithm could not be tuned both at compile time and at run-time. Over the last decade, advancements in process technology enable multimillions of transistors to be fitted onto a silicon chip. This allows for more complex functions to be implemented on FPGAs. This riding curve of the process technology emerges the use of FPGAs in scientific computing including in sequence alignment. This hardware acceleration platform capable to implement parallel processing as the special purpose architecture with added convenient of re-programmability. Consequently, tremendous FPGA implementations of both DNA and protein sequence alignment with optimal alignment have been reported includes in [9],[10],[11],[12] and [13]. However, aligning sequences (e.g. biological) that are often hundreds if not thousands-residue long requires considerable logic resources. This is a challenge even in modern FPGAs. In most of the reported FPGA implementations, query sequences are compared against database sequence on single silicon chip by partitioning the S-W algorithm into smaller alignment steps and process them sequentially in multiple passes over the same systolic array, the so-called folding technique. The reported folded S-W implementations in [9] and [11] for instance, require *n* configuration elements (henceforth referred to as CE) in each processing elements (PE) to hold *n* substitution matrix columns, each equivalent to one query residue held by the PE. In addition to the *nCE* area overhead, this results in *n*-fold increase in the configuration time of the PEs. Another method which is reported in [13] used run time reconfiguration (RTR) technique as an approach to configure the PEs on the fly between folds or passes. However, due to the limited reconfiguration bandwidth and extra logic resources needed for PE reconfiguration, the resulting overhead is still considerable. In this paper, we propose a novel sequence alignment core, which based on the widely used double-buffering or ping-pong technique. In this proposed architecture, it is referred to as Overlapped Computation and Configuration (OCC). This approach efficiently schedules between alignment matrix computation and CE configuration for subsequent fold computations with the following added advantages over other typical folded S-W techniques;

Optimize Space Complexity: It alternately uses a fixed number of CEs (equal to 2) to compute any length of query sequences. This will optimize logic resources

instead of replicating look-up tables in the PE to align longer query

Optimize Time complexity: It optimizes the total execution time by virtually removing the configuration time overhead through the overlapping of alignment matrix computation and CE configuration.

The following section discusses the Smith Waterman algorithm and the GOTOH algorithm with linear and affine gap penalty respectively. The efficient scheduling strategy in pairwise biological sequence alignment is discussed in section III. Section IV details the proposed sequence alignment core architecture to implement the efficient scheduling technique for the case of GOTOH algorithm. The corresponding results of the novel architecture which is implemented on hardware will be presented in section V before the conclusion is laid out in section VI

II. ALIGNING SEQUENCES WITH OPTIMAL RESULTS

Biological sequences diverges from a common ancestor due to the process of mutation, selection and random genetic drift [1]. Mutation for instance, involves three main processes: Substitution of residues, insertion of new residues and deletion of existing residues. Both insertion and deletion are sometimes referred to as gaps. Gaps in alignment are undesirable and thus penalized. The cost of gaps depends on its length and generally, there are two different ways to penalize gaps; linear and affine gap penalties. Equation (1) shows the Smith-Waterman algorithm with a linear gap penalty. It is introduced in 1981 by T. F. Smith and M. S. Waterman. Given a query sequence, X = $x_1,x_2,x_3..x_i...x_M$ (of length M) and Y = $y_1,y_2,y_3..y_j...y_N$ (of length N), this DP-based alignment algorithm searches for the best alignment between sub-sequences of *x* and *y* using matrix *M(i,j)*. This matrix calculates the largest score among the four alternatives and it is built recursively using (1).

$$M(i,j) = \max\begin{cases} 0 \\ M(i-1,j-1) + s(x_i, y_j) \\ M(i-1,j) - d \\ M(i,j-1) - d \end{cases} \quad (1)$$

The $s(x_i,y_j)$ is i_{th} and j_{th} of residue *x* and *y* in a two dimensional matrix. It represents probabilistic score which describes biological relationship of amino acids x_i and y_j as shown in the substitution matrix in Fig. 1. *BLOSUM62* and *PAM* are other examples of amino acids probabilistic models. Fig. 1 presents an example of the *BLOSUM50* substitution matrix[1]. The entries on the main diagonal as highlighted in bold represent identical residue pairs. The 20x20 matrix comprises of 20 amino-acids (or residues).

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	**5**	−2	−1	−2	−1	−1	−1	0	−2	−1	−2	−1	−1	−3	−1	1	0	−3	−2	0
R	−2	**7**	−1	−2	−4	1	0	−3	0	−4	−3	3	−2	−3	−3	−1	−1	−3	−1	−3
N	−1	−1	**7**	2	−2	0	0	0	1	−3	−4	0	−2	−4	−2	1	0	−4	−2	−3
D	−2	−2	2	**8**	−4	0	2	−1	−1	−4	−4	−1	−4	−5	−1	0	−1	−5	−3	−4
C	−1	−4	−2	−4	**13**	−3	−3	−3	−3	−2	−2	−3	−2	−2	−4	−1	−1	−5	−3	−1
Q	−1	1	0	0	−3	**7**	2	−2	1	−3	−2	2	0	−4	−1	0	−1	−1	−1	−3
E	−1	0	0	2	−3	2	**6**	−3	0	−4	−3	1	−2	−3	−1	−1	−1	−3	−2	−3
G	0	−3	0	−1	−3	−2	−3	**8**	−2	−4	−4	−2	−3	−4	−2	0	−2	−3	−3	−4
H	−2	0	1	−1	−3	1	0	−2	**10**	−4	−3	0	−1	−1	−2	−1	−2	−3	2	−4
I	−1	−4	−3	−4	−2	−3	−4	−4	−4	**5**	2	−3	2	0	−3	−3	−1	−3	−1	4
L	−2	−3	−4	−4	−2	−2	−3	−4	−3	2	**5**	−3	3	1	−4	−3	−1	−2	−1	1
K	−1	3	0	−1	−3	2	1	−2	0	−3	−3	**6**	−2	−4	−1	0	−1	−3	−2	−3
M	−1	−2	−2	−4	−2	0	−2	−3	−1	2	3	−2	**7**	0	−3	−2	−1	−1	0	1
F	−3	−3	−4	−5	−2	−4	−3	−4	−1	0	1	−4	0	**8**	−4	−3	−2	1	4	−1
P	−1	−3	−2	−1	−4	−1	−1	−2	−2	−3	−4	−1	−3	−4	**10**	−1	−1	−4	−3	−3
S	1	−1	1	0	−1	0	−1	0	−1	−3	−3	0	−2	−3	−1	**5**	2	−4	−2	−2
T	0	−1	0	−1	−1	−1	−1	−2	−2	−1	−1	−1	−1	−2	−1	2	**5**	−3	−2	0
W	−3	−3	−4	−5	−5	−1	−3	−3	−3	−3	−2	−3	−1	1	−4	−4	−3	**15**	2	−3
Y	−2	−1	−2	−3	−3	−1	−2	−3	2	−1	−1	−2	0	4	−3	−2	−2	2	**8**	−1
V	0	−3	−3	−4	−1	−3	−3	−4	−4	4	1	−3	1	−1	−3	−2	0	−3	−1	**5**

Fig. 1 The BLOSUM50 substitution matrix [1]

In linear gap penalty in (1), *d* is a constant penalty and it penalizes gaps of length *g* linearly i.e *penalty(g)=-g*d*. A more efficient gap penalty is introduced by GOTOH [14] in 1982, which is referred to as affine gap penalty as shown in (2). In this type of gap penalty, a constant gap cost is given when opening a new gap (gap opening or *d*), while a linear and often smaller gap penalty is given for subsequent gap extensions (*e*) i.e. *penalty(g)=-d-(g-1)*e*. The *M(i,j)* is the score up to *(i,j)* where residue x_i is aligned to residue y_j. The $I_x(i,j)$ is the best score, where residue x_i is aligned to a gap and finally the $I_y(i,j)$ is the best score, where residue y_i is aligned to a gap. In this paper, we will implement Eq. (2) and details of its corresponding hardware architecture are discussed in the following section.

$$M(i,j) = \max\begin{cases} M(i-1,j-1) + s(x_i, y_j) \\ I_x(i-1,j-1) + s(x_i, y_j) \\ I_y(i-1,j-1) + s(x_i, y_j) \end{cases}$$
$$I_x(i,j) = \max\begin{cases} M(i-1,j) - d \\ I_x(i-1,j) - e \end{cases} \quad (2)$$
$$I_y(i,j) = \max\begin{cases} M(i,j-1) - d \\ I_y(i,j-1) - e \end{cases}$$

III. THE EFFICIENT SCHEDULING TECHNIQUE

The proposed core is useful in processing long sequences whereby it is not possible to allocate enough PEs on the available FPGA device. Rather than replicating PE-related substitution matrix coefficients at each pass (or fold), our technique uses a fixed number of configuration elements (equal to two as shown in Fig. 2(a), namely CE_0 and CE_1) regardless of the folding factor. Alignment matrix computation uses one CE (CE_0) while configuring the content of another element (CE_1) for the subsequent pass (or fold), and vice-versa in the subsequent pass. In the

example shown in Fig. 2(b), a folding factor of four is assumed i.e. we want to align a query sequence of length $4N_{PE}$ and we can only implement N_{PE} in hardware.

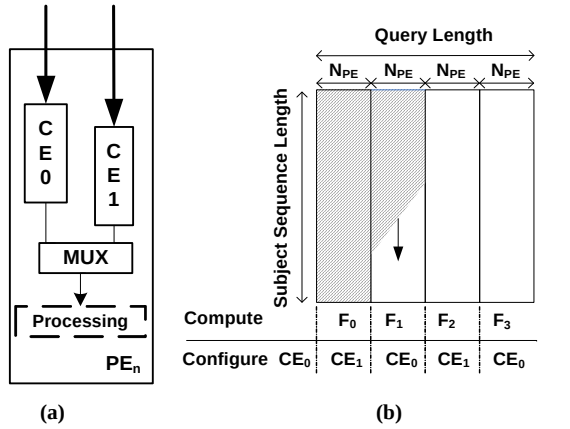


Fig. 2 (a) Internal PE structure with two CEs (b) Computation and configuration over the same systolic array

The passes or folds are denoted as F_0, F_1, F_2 and F_3 in Fig. 2(b). To allow efficient scheduling between configuration and computation, all CE_0 elements in the PEs are updated during the *Initial Config*. phase as depicted in Fig. 3(a).

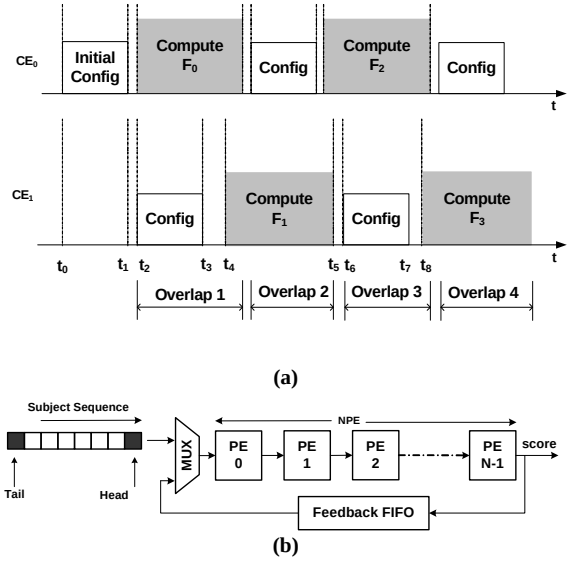


Fig. 3 (a) The efficient scheduling technique between CE configuration and alignment matrix computation. (b) Subject sequence flows through processing elements of length N_{PE} with multiple-pass computation over the same PE systolic arrays.

Once the first computation starts (F_0), CE_1 of all PEs are updated with new coefficients for the next fold computation (F_1) (labelled as *Overlap 1*). F_1 may starts computation at t_4 when the tail of the current subject sequence leaves the first PE. During F_1 alignment computations, F_0 finishes its task (once the tail of the current subject sequence (see Fig 3(b)) leaves the last PE) and the second overlap operation (*Overlap 2*) occurs, where CE_0 will be updated with new coefficients for subsequent fold computation (F_2). This overlapped operation and configuration continues until the last fold *Overlap 4* in Fig. 3(a)). Note that, during each last computation fold, CE_0 is configured with new coefficients for the subsequent subject sequence in the database until all subject sequences in the database are exhausted.

IV. THE NOVEL HARDWARE ARCHITECTURE

This section discusses three novel architectures i.e. the *PE*, the *PARALLEL LOADER* and the *OCC Scheduler* (*MAIN CONTROLLER)*, which are designed to implement the efficient scheduling technique mentioned in section III. The overall system architecture of the biological sequence alignment core with the proposed architectures is shown in Fig. 4. Note that, all computational parameters include the data width (*dw*) and the compute data width (*cdw*) are parameterisable. The PE performs 'processing' tasks i.e. computes the elementary functions of the DP algorithm and communicates with the next PE using regular interconnections to form a linear systolic array of PEs (*PE_BLOCK*). Unlike typical folded Smith-Waterman implementations, the proposed PE has only two CEs, thus proper scheduling is required to alternately use CE_0 and CE_1 for configuration and alignment matrix computation. This way, a CE is configured with different probability score tables at different folds while another CE holds a column of substitution matrix scores for the corresponding fold computation. This enables efficient use of logic resources. Moreover, with the efficient scheduling technique, the overall system throughput increases significantly. To implement the OCC technique, the *MAIN CONTROLLER* schedules both the configuration and the computation modes to run simultaneously. This operation virtually removes CE configuration time at every fold computation. Another instance which is crucial for the efficient scheduling technique is the *PARALLEL LOADER*. As it name implies the loader is designed to configure CEs in parallel regardless the number of PEs or length of query sequence. This way, the time to configure CE in all PEs is less than the time elapsed to compute the alignment matrix. This enables smooth scheduling of the concurrent operations (alignment matrix computation and CE configuration) during each fold computation. Details regarding the three novel architectures are discussed in the following sub-sections.

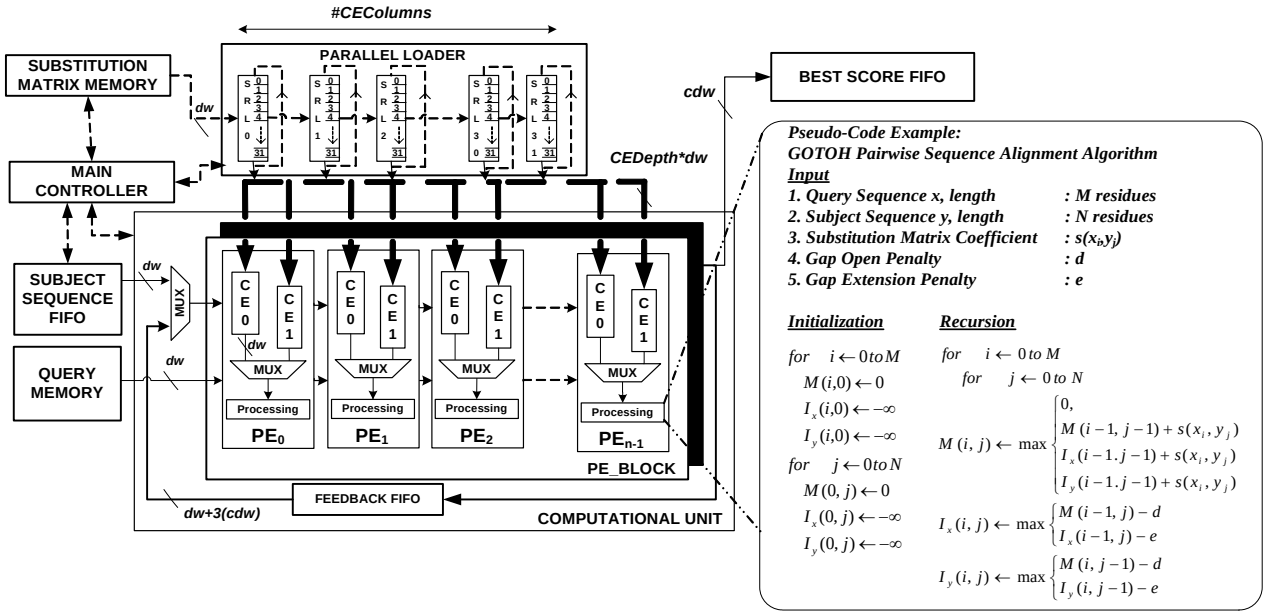


Fig. 4 The biological sequence alignment core architecture

A) The Internal PE Architecture

The inner structure of the PE, which implements the GOTOH local alignment algorithm [14] is illustrated in Fig.5. The PE is designed with all its computational parameters includes the gap data width (*gdw*) and the depth of the CE (*CEDepth*) are parameterisable. In this architecture, the *gdw* is four bit, which is enough to represent the gap open and gap extension penalty scores for the affine gap function. On the other hand, the *CEDepth* is set to 32 elements (5 bits), which suffices for DNA (nucleotides of A, G, T and C) and protein (20 amino-acids) sequences and it maps very well to Xilinx FPGA slice architecture. The main task of the PE is implementing the 'processing' operation. The affine gap penalty PE consists of three arithmetic units, i.e. the best score ($M(i, j)$) of residue x_i and y_j, the best score of insertion with respect to x direction ($I_x(i, j)$) of residue x_i and y_j and the best score with respect to y direction ($I_y(i, j)$) of residue x_i and y_j. All of these units are from the top down to the bottom of the shaded boxes respectively. The $M(i, j)$) unit determines the highest score among the three alternatives in the PE at each processing step. The *Cfg* input is added to the maximum expression to tackle different types of alignment. In this architecture, *Cfg* is set to '0'as it implements the local alignment algorithm, (i.e. alignment scores saturated to zero). For the case of global alignment, the *Cfg* input is set to minus infinity. The *PE Best Score* unit calculates the current PE's best score. Then, it propagates the score (maximum-so-far) to subsequent PE in a chain across the PE systolic arrays. If the accumulated score satisfied a given threshold value, the best score of the last PE with its corresponding subject sequence address are stored in the *Best Score FIFO*, otherwise the score and the

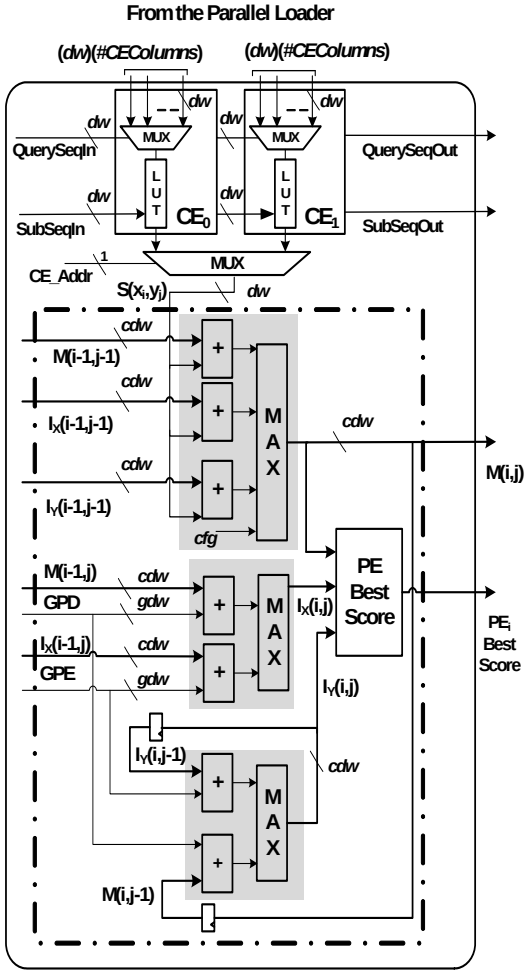


Fig. 5 Internal PE architecture for the GOTOH algorithm

subject sequence are disregarded. As mentioned earlier, both configuration and computation modes run simultaneously except for *Initial Config*. mode. For the sake of clarity, each mode is explained separately in this section. During the configuration mode, each of the query residue flows through the *QuerySeqIn* port to fetch its corresponding substitution matrix column in turn, where each column of substitution matrix array (*CEDepth x #CEColumns*) for the case of this implementation, consists of size 32 *x* 32 elements.

During the computation mode, the *CE_Addr* port selects coefficients either from CE_0 or CE_1 depending on the current number of fold computation. For all even-numbered fold computations, CE_0 supplies its substitution matrix scores for PE computation (during this fold, CE_1 is configured with new coefficients for subsequent fold computation). Similarly, during all odd-numbered computation, CE_1 supplies its coefficients for alignment matrix computation. During this fold, CE_0 is configured with new coefficients for subsequent fold computation. This operation continues following the arrangements as discussed in section III. In either fold computation, alignment matrix is calculated as subject sequence residue flows through the *SubSeqIn* input to fetches its corresponding substitution matrix coefficient, $s(x_i,y_j)$ for the alignment matrix computation.

B) The Parallel Loader

Fig. 6 illustrates the internal elements of the CE loader. Both the *CEDepth* and the *#CEColumns* are parameterisable and for this implementation both are set to 32 elements.

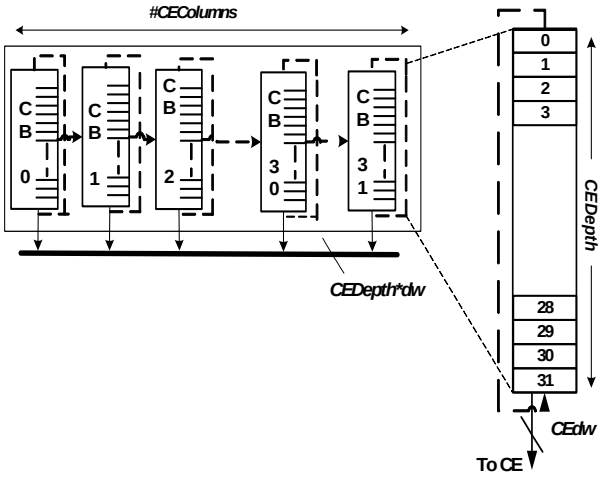


Fig. 6 The Parallel Loader with the Circular Buffers

The loader is made up of circular buffers (*CBx*), which are implemented efficiently using shift registers based on the FPGA's Look-up Tables (LUTs), referred to as SRL32 [15]. These buffers constantly revolve the columns of the substitution matrix, presenting complete column's elements for all of the CEs at every multiple of 32 cycles to the pipeline PEs (one column equals to *CEDepth*, which is 32 elements for the case of 5-bit input). Thus, the worst case configuration time for all CEs is *2 X CEDepth* clock cycles. The data width or *dw* is also parameterisable and for the case of the Blosum50, 5-bit is enough to represent the probability scores.

C) The OCC Scheduler

The simplified state machine in Fig. 7 illustrates the overall operations of the scheduler, which is designed in the *MAIN CONTROLLER* of the proposed core. This scheduler manages the fixed CE resources in the PE by implementing the efficient scheduling technique as discussed in section III.

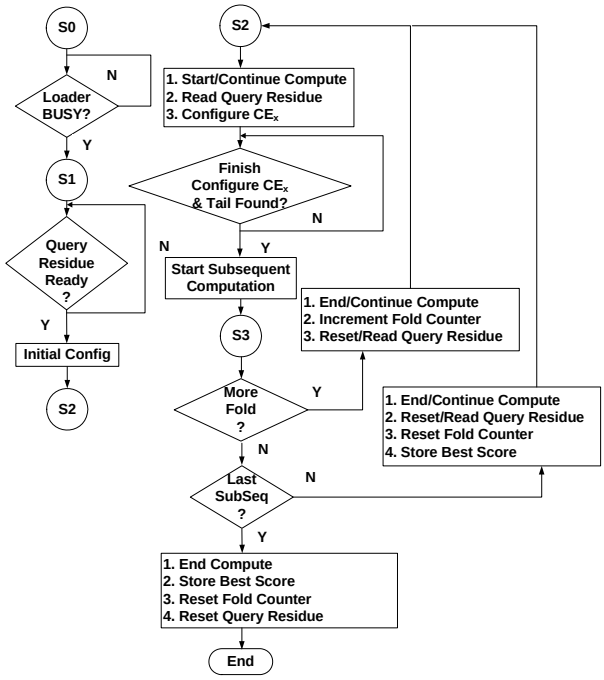


Fig. 7 Simplified state machines of the OCC Scheduler

State *S0* involves configurations of all memory-based units including the *QUERY MEMORY*, the *SUBSTITUTION MATRIX MEMORY* and the *PARALLEL LOADER*. Once all of these units are ready, the next state, i.e. *S1*, initiates CE_0 configuration (*Initial Config*. phase). Beginning from this state, the query sequence is partitioned into several sub-sequences depending on the number of folds and each query sub-sequence is read separately. This way, the corresponding CE could be configured efficiently with only the CE-related query residues are read from the *QUERY MEMORY* during configuration. The overlapped operation occurs during state *S2*, whereby both computation and configuration occur simultaneously. If the current configuration has finished and the pipeline is ready for subsequent fold computation (typically CE configuration finishes earlier than the current computation), then the state machine moves to *S3*.

This state decides either to continue with the subsequent overlap operation (by incrementing the fold counter) or reset it (if the fold counter reaches it maximum fold) to align next subject sequence in the database. At each processing pass, the *FEEDBACK FIFO* stores intermediate results before these are fed back to the input of the PE systolic arrays for subsequent fold processing. For every subject sequence that passes through the pipeline, the controller triggers the *BEST SCORE FIFO* to save the best score of the subject sequence if the best score of the last of the last PE is satisfied a given threshold value.

V. IMPLEMENTATION RESULTS

This section discusses performance evaluation of the proposed core in two ways. The first compares the novel architecture against the state-of-the-art hardware implementation reported in [11]. The second compares the proposed architecture against the widely used software implementation of local alignment, namely SSEARCH35, as well as other FPGA implementations, taking into account all input/output overheads. As a sample, both architectures are tested with query sequences ranging from 128 residues to 2048 residues from the protein knowledgebase (UniProtKB), as shown in Table 1, to evaluate the speed-up improvement with respect to the area complexity of the proposed core. Each of the query sequences is aligned against different lengths of subject sequences (16 up to 1024 residues) in a systolic array of 128PEs with different folds. The base implementation reported in [11] has *n*CEs (where n equal to the number of folds), while the proposed core has only two. Thus, normalization is required in order to evaluate the speed-up performance of both cores fairly. We normalize the execution time results shown in Table 1 by the standard Xilinx logic cell (LC) unit, which is an abstract logic resource measure independent from the particular FPGA family's slice architectures [16].

TABLE 1

EXECUTION TIME AND AREA NORMALIZED SPEED-UP OF THE PROPOSED CORE

Query Accession (length)	Q2IJ63 (128)	Q96B36 (256)	Q16515 (512)	Q96RT8 (1024)	Q8IYD8 (2048)
#Folds	1	2	4	8	16
Ref [11] (*us*)	59.64	120.71	242.81	486.99	975.36
Proposed (*us*)	40.00	78.38	155.1	310.24	622.64
Speed-up[a]	1.49	1.54	1.57	1.57	1.57
#LCs Proposed	98912	98912	98912	98912	98912
#LCs Ref[11]	93780	96852	102996	115284	139860
Area Ratio[b]	1.05	1.02	0.96	0.86	0.71
Area Normalized Speed-up [c]	1.42	1.51	1.64	1.83	2.21

An affine gap PE comprises ~117 logic slices (468 LCs, i.e. 4LCs/slices), while *FEEDBACK FIFO* consumes 54Kb of BRAM. To take into account the FIFO logic resources, we synthesized the *FEEDBACK FIFO* and the PE elements using Cadence Build Gates (2005) with 0.18um UMC process technology, and noted the gate equivalent of each. By dividing the gate count results, we found that 1 Kbit of memory (BRAM) consumes the equivalent of 18 LCs. This allows us to normalize the speed-up with all area results (logic and memory) in terms of LCs. The area normalized speed-up (speed-up/logic cell) in Table 1 demonstrates that the proposed architecture has a normalized speed-up higher than 40 percent, growing linearly with the number of folds. The proposed core was implemented on Alpha Data ADM-XRC-5LX card with Virtex5vlx110 FPGA on it. A database sequence (release 2012_06 of 13-Jun-2012) from UniProtKB/TrEMBL comprises of 22,660,469 sequence entries with a total of 7,407,531,063 amino acids is used in this analysis. It consumes 2.36 GB of memory, which is stored in the host memory and transferred through the PCI bus with data transfer rate of 2,112 Mbps.

TABLE 2

TOTAL EXECUTION TIME (CLOCKED AT 100MHZ) AND SPEED-UP OF THE PROPOSED CORE AGAINST THE SSEARCH35

Query Accession	Length	# Fold	Total Execution Time(s)		Speed up
			Ours	SSEARCH	
P02652	100	1	91	9416	103.32
Q9H3V2	200	2	152	17160	113.00
Q8NC42	400	4	303	35992	118.70
A6NGE4	600	6	451	55704	123.50
B3KY11	800	8	599	74888	125.00
A8KA62	1000	10	766	96888	126.50
Q8NEL9	1200	12	878	112200	127.80
B2RNT9	1400	14	1067	137280	128.60
D3DNT2	1600	16	1215	157696	129.80
Q9BYP7	1800	18	1370	182864	133.50
Q12873	2000	20	1512	211024	139.60

Table 2 summarizes the overall core performance with varying fold factors against SSEARCH35, which runs on Intel(R) Quad Core 64-bit Q8300 (processor speed of 2.50 GHz and RAM of 4.00GB). The speed-up is calculated by dividing the software execution time with the total execution time of the proposed core. From the last column in Table 2, it clearly shows that the speed-up of the proposed core with efficient scheduling technique grows linearly.

Performing fair and meaningful comparisons against different FPGA implementations is difficult due to different types of devices and families used. This has led us to use the LCs as a normalization factor to provide fair evaluation. Here in (3), an independent performance evaluator (normalized speed-up/logic cell/process technology) is proposed to

effectively compare the core performance against various FPGA implementations.

$$SpeedUp_{normalized} = \frac{SpeedUp}{LC} \times LUT\ Delay \qquad (3)$$

Where, LUT Delay is FPGA basic look-up table delay, which varies depending on the device's process technology, among other factors. We normalize the speed-up figures with the area consumption by dividing the speed-up with the ratio of logic cells (LC-equivalent) consumed by each implementation (see Table 3 column [B]).

TABLE 3
NORMALIZED SPEED-UP PERFORMANCE (FOLD OF 12)

Reference & Device	Total LCs Used	Area Ratio[1]	LUT Delay Ratio[2]	Speed-up Proposed[3] (OCC) vs. Ref.		
				A	B	C
[9] XC2V6000	31,059	0.69	0.23	4.58	6.62	1.53
[11] XC5VLX110	37,807	0.57	1.00	1.15	2.03	2.03
[13] XC2V6000	43,546	0.49	0.23	5.13	10.40	2.40

[1]LCs consumed by our proposed core / LCs consumed by each Ref.
[2] LUT delay of the XC5VLX110 FPGA / LUT delay of each Ref.
[3]the OCC core utilizes 21,458 LCs for maximum PEs of 140.
[A] = Execution time of each Ref. /Execution time of the proposed core
[B] = Area normalized speed-up
[C] = Normalized speed-up per area per process technology

In addition, we normalize with the fabrication technology by multiplying the speed-up figures with the ratio of basic LUT delays of the FPGA technologies used (see Table 3 column [C]). After doing this, Table 3 clearly shows that the proposed core is the most efficient. Note that in other cases [17] and [18] the normalized speed-up performance could not be determined due to limited information provided, which shows the need for a standard common experimental reporting framework.

VI. CONCLUSION

The proposed architecture has successfully optimized the execution time of DP-based pairwise sequence alignment algorithms in hardware through efficient scheduling of alignment matrix computation and substitution matrix pre-loading for subsequent computations. A new performance metric normalized with area and process technology (speed-up/logic cell/process technology), has been proposed to independently compare the proposed core against other FPGA implementations. Results show the core achieves over 40 percent normalized speed-up compared to the state-of-the-art, with the speed-up growing linearly with the number of folds e.g. reaching 120 percent for 16-fold improvement. Moreover, comparison with its corresponding software implementation shows that the speed-up grows linearly with the fold factor (140x speed-up for fold of 20) with fixed logic resources and allows changing of fold factors at run time.

VII. REFERENCES

[1] R. Durbin, Eddy, S., Krogh, A., Mitchison, G, Biological Sequence Analysis: Probabilistic Models for Proteins and Nucleic Acids: Cambridge University Press, Cambridge UK, 1998.

[2] D. T. Hoang, "FPGA Implementation of Systolic Sequence Alignment," presented at International Workshop on Field Programmable Logic and Applications, Vienna, Austria, 1992.

[3] M. Borah, R. S. Bajwa, S. Hannenhalli, and M. J. Irwin, "A SIMD solution to the sequence comparison problem on the MGAP," presented at International Conference on Application Specific Array Processors, 1994.

[4] L. G. David M. Dahle, Eric Rice, Richard Hughey, "The UCSC Kestrel General Purpose Parallel Processor.," presented at PDPTA, 1999.

[5] B. Schmidt, H. Schroder, and M. Schimmler, "Massively parallel solutions for molecular sequence analysis," presented at International Symposium on Parallel and Distributed Processing (IPDPS 2002), 2002.

[6] P. Guerdoux-Jamet and D. Lavenier, "SAMBA: hardware accelerator for biological sequence comparison," Journal of Computer applications in the biosciences : CABIOS, vol. 13, pp. 609-615, 1997.

[7] R. K. Singh, D. L. Hoffman, S. G. Tell, and C. T. White, "BioSCAN: a network sharable computational resource for searching biosequence databases," Computer applications in the biosciences : CABIOS, vol. 12, pp. 191-196, 1996.

[8] E. Chow, T. Hunkapiller, J. Peterson, and M. S. Waterman, "Biological information signal processor," presented at Application Specific Array Processors, 1991. Proceedings of the International Conference on, 1991.

[9] K. Benkrid, L. Ying, and A. Benkrid, "A Highly Parameterized and Efficient FPGA-Based Skeleton for Pairwise Biological Sequence Alignment," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 17, pp. 561-570, 2009.

[10] P. Zhang, G. Tan, and G. R. Gao, "Implementation of the Smith-Waterman Algorithm on A Reconfigurable Supercomputing Platform," Altera Corporation 2007.

[11] M. N. Isa, K. Benkrid, T. Clayton, C. Ling, and A. T. Erdogan, "An FPGA-based parameterised and scalable optimal solutions for pairwise biological sequence analysis," presented at 2011 NASA/ESA Conference on Adaptive Hardware and Systems (AHS), 2011.

[12] Y. Yoshiki, M. Yosuke, M. Tsutomu, and K. Akihiko, "High Speed Homology Search Using Run-Time Reconfiguration," in Proceedings of the Reconfigurable Computing Is Going Mainstream, 12th International Conference on Field-Programmable Logic and Applications: Springer-Verlag, 2002.

[13] T. F. Oliver, B. Schmidt, and D. L. Maskell, "Reconfigurable architectures for bio-sequence database scanning on FPGAs," IEEE Transactions on Circuits and Systems II, vol. 52, pp. 851-855, 2005.

[14] G. Osamu, "An improved algorithm for matching biological sequences," Journal of Molecular Biology, vol. 162, pp. 705-708, 1982.

[15] "Virtex-5 Family User Guide," Xilinx, Inc., San Jose, CA 2009.

[16] "Virtex-5 Family Overview," Xilinx, Inc., San Jose, CA 2009.

[17] Y. Yamaguchi, et al., "FPGA-Based Smith-Waterman Algorithm: Analysis and Novel Design Reconfigurable Computing: Architectures, Tools and Applications," vol. 6578, Lecture Notes in Computer Science: Springer Berlin / Heidelberg, 2011, pp. 181-192.

[18] J. Xianyang, L. Xinchun, X. Lin, Z. Peiheng, and S. Ninghui, "A Reconfigurable Accelerator for Smith Waterman Algorithm," IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 54, pp. 1077-1081, 2007.

References

[1] N. M. Luscombe, D. Greenbaum, and M. Gerstein, "What is bioinformatics? An introduction and overview," *Yearbook of Medical Informatics 2*, pp. 83, 2001.

[2] E. B. Institute, "Amino Acids Abbreviation Table," in *http://www.ebi.ac.uk/2can/biology/molecules_small_aatable.html [Last accessed: Jan 2013]*.

[3] R. Durbin, Eddy, S., Krogh, A., Mitchison, G, "Biological Sequence Analysis: Probabilistic Models for Proteins and Nucleic Acids," pp. 2-11, 1998.

[4] N. M. Luscombe, D. Greenbaum, and M. Gerstein, "What is bioinformatics? An introduction and overview," 2001.

[5] NCBI, "GenBank, RefSeq, TPA and UniProt: What's in a Name? ," in *http://www.ncbi.nlm.nih.gov/projects/RefSeq/GenBankvsRefSeq.pdf [Last accessed: October 2012]*.

[6] D. A. Benson, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell, and E. W. Sayers, "GenBank," *Nucleic Acids Research*, vol. 39, pp. D32-D37, 2011.

[7] UniProt, "Why is UniProtKB composed of 2 sections, UniProtKB/Swiss-Prot and UniProtKB/TrEMBL?," in *http://www.uniprot.org/faq/7 [Last accessed : March 2013]*.

[8] UniProtKB/Swiss-Prot, "UniProtKB/Swiss-Prot protein knowledgebase release 2012_09," in *http://www.uniprot.org/news/2012/10/03/release [Last accessed : December 2012]*.

[9] M. S. Rosenberg, "Sequence Alignment; Methods, Models, Concepts, and Strategies," pp. 3-4, 2011.

[10] M. O. Dayhoff, R. M. Schwartz, and B. C. Orcutt, "A model of evolutionary change in proteins," *Atlas of Protein Sequence and Structure*, vol. 5, pp. 345 - 352, 1978.

[11] S. Henikoff and J. Henikoff, "Amino acid substitution matrices from protein blocks," *Proc Natl Acad Sci USA*, vol. 89, pp. 10915 - 10919, 1992.

[12] S. R. Eddy, "Where Did the BLOSUM62 Alignment Score Matrix Come From?," *Nature Biotechnology*, vol. 22, pp. 1035-1036, 2004.

[13] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of Molecular Biology*, vol. 48, pp. 443-453, 1970.

[14] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *J Mol Biol*, vol. 147, pp. 195 - 197, 1981.

[15] G. Osamu, "An improved algorithm for matching biological sequences," *Journal of Molecular Biology*, vol. 162, pp. 705-708, 1982.

[16] D. J. Lipman and W. R. Pearson, "Rapid and sensitive protein similarity searches," *Science. 1985 Mar 22;227(4693):1435-41.*, 1985.

[17] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of Molecular Biology*, vol. 215, pp. 403-410, 1990.

[18] W. R. Pearson and D. J. Lipman, "Improved Tools for Biological Sequence Comparison," *National Academy of Sciences of the United States of America*, vol. 85, pp. 2444-2448, 1988.

[19] S. R. Eddy, "Hidden Markov models," *Current opinion in structural biology*, vol. 6, pp. 361-365, 1996.

[20] S.R.Eddy, "What is Hidden Markov Model?," *Journal of Computational Biology*, vol. 22, pp. 1315-1316, 2004.

[21] S. R. Eddy, "Multiple alignment using hidden Markov models," *Proc Int Conf Intell Syst Mol Biol. 1995;3:114-20.*, vol. 3, pp. 114-120, 1995.

[22] S. Brown and J. Rose, "Architecture of FPGAs and CPLDs: A Tutorial," *IEEE Design and Test of Computers*, vol. 13, pp. 42-57, 1996.

[23] J. Rose, R. J. Francis, D. Lewis, and P. Chow, "Architecture of field-programmable gate arrays: the effect of logic block functionality on area efficiency " *IEEE Journal of Solid-State Circuits*, vol. 25, pp. 1217-1225, 1990.

[24] I. Kuon, R. Tessier, and J. Rose, "FPGA Architecture: Survey and Challenges," vol. 2, pp. 136-235, 2008.

[25] Xilinx, "7 Series FPGAs Overview, DS180 (v1.13)," 2012.

[26] M. A. Tahir, "Hardware and Software Solutions for Prostate Cancer Classification using Multispectral Images," in *Faculty of Enginneering*, vol. PhD. Belfast, UK: Queen's University, Belfast, UK, 2006, pp. 178.

[27] Xilinx, "Virtex-II Platform FPGAs:Complete Data Sheet, DS031(v3.4)," 2007.

[28] Xilinx, "Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Introduction and Overview," 2011.

[29] Xilinx, "Virtex-4 Family Overview, DS112 (v3.1)," 2010.

[30] Xilinx, "Virtex-5 FPGA User Guide, ug190," 2008.

[31] Xilinx, "Virtex-6 Family Overview," 2012.

[32] Xilinx, "Advantages of the Virtex-5 FPGA 6-Input LUT Architecture, WP284," 2007.

[33] C. M. Maxfield, *The Design Warrior's Guide to FPGAs,: Devices, Tools and Flows*: Elsevier, 2004.

[34] Xilinx, "LogiCORE IP MicroBlaze Micro Controller System (v1.1), DS865," 2012.

[35] Xilinx, "FPGA Design Flow Overview," 2008.

[36] K. Tatas, K. Siozios, and D. Soudris, "A Survey of Existing Fine-Grain Reconfigurable Architectures and CAD tools," in *Fine- and Coarse-Grain Reconfigurable Computing*: Springer Netherlands, 2008, pp. 3-87.

[37] Xilinx, "Partial Reconfiguration User Guide," 2010.

[38] Xilinx, "Partial Reconfiguration User Guide, UG702," 2012.

[39] M. Novati, "2D Relocation of Self Dynamical Run-Time Reconfiguration," 2007.

[40] T. F. Oliver, B. Schmidt, and D. L. Maskell, "Reconfigurable architectures for bio-sequence database scanning on FPGAs," *IEEE Transactions on Circuits and Systems II*, vol. 52, pp. 851-855, 2005.

[41] A. Data, "The ADM-XRC-5LX PCI Mezzanine Card," 2009.

[42] Xilinx, "Virtex-5 LX Platform Overview, DS100 (v1.1)," 2006.

[43] E. SR., "What Is Dynamic Programming?," in *Magazine of Nature Biotechnology*, vol. 22, 2004, pp. 909-910.

[44] O. Gotoh, "An improved algorithm for matching biological sequences," *J Mol Biol*, vol. 162, pp. 705 - 708, 1982.

[45] H. T. Kung and C. E. Leiserson, *Systolic Arrays for (VLSI)*: Carnegie-Mellon University, Department of Computer Science, 1978.

[46] M. J. Foster and H. T. Kung, "The Design of Special-Purpose VLSI Chips," *Computer*, vol. 13, pp. 26-40, 1980.

[47] H. T. Kung, "Systolic (VLSI) arrays for relational database operations," presented at international conference on Management of data (SIGMOD '80) New York, USA, 1980.

[48] D. P. Lopresti, "P-NAC: A Systolic Array for Comparing Nucleic Acid Sequences," *Computer*, vol. 20, pp. 98-99, 1987.

[49] D. T. Hoang, "FPGA Implementation of Systolic Sequence Alignment," presented at International Workshop on Field Programmable Logic and Applications, Vienna, Austria, 1992.

[50] M. Borah, R. S. Bajwa, S. Hannenhalli, and M. J. Irwin, "A SIMD solution to the sequence comparison problem on the MGAP," presented at International Conference on Application Specific Array Processors, 1994.

[51] L. G. David M. Dahle, Eric Rice, Richard Hughey, "The UCSC Kestrel General Purpose Parallel Processor.," presented at PDPTA, 1999.

[52] B. Schmidt, H. Schroder, and M. Schimmler, "Massively parallel solutions for molecular sequence analysis," presented at International Symposium on Parallel and Distributed Processing (IPDPS 2002), 2002.

[53] E. Chow, T. Hunkapiller, J. Peterson, and M. S. Waterman, "Biological information signal processor," presented at Proceedings of the International Conference on Application Specific Array Processors (ASAP), 1991.

[54] R. K. Singh, D. L. Hoffman, S. G. Tell, and C. T. White, "BioSCAN: a network sharable computational resource for searching biosequence databases," *Computer applications in the biosciences : CABIOS*, vol. 12, pp. 191-196, 1996.

[55] P. Guerdoux-Jamet and D. Lavenier, "SAMBA: hardware accelerator for biological sequence comparison," *Journal of Computer applications in the biosciences : CABIOS*, vol. 13, pp. 609-615, 1997.

[56] D. A. Benson, et al., "GenBank," *Nucleic Acids Research*, vol. 28, pp. 15-18, 2000.

[57] Y. Yamaguchi, Maruyama, T., and Konagaya,A., "High Speed Homology Search with FPGAs," in *The Pacific Symposium on Biocomputing*, 2002, pp. 271-282.

[58] R. P. Jacobi, M. Ayala-Rincon, L. G. Carvalho, C. H. Llanos, and R. W. Hartenstein, "Reconfigurable systems for sequence alignment and for general dynamic programming," *Genet Mol Res. 2005 Sep 30;4(3):543-52.*, 2005.

[59] T. Van Court and M. C. Herbordt, "Families of FPGA-based accelerators for approximate string matching," *Microprocessors and Microsystems*, vol. 31, pp. 135-145, 2007.

[60] A. Mohamed, E.-A. Esam, and T. Mohamed, "DNA and Protein Sequence Alignment with High Performance Reconfigurable Systems," in *the Second NASA/ESA Conference on Adaptive Hardware and Systems*: IEEE Computer Society, 2007.

[61] K. Benkrid, L. Ying, and A. Benkrid, "A Highly Parameterized and Efficient FPGA-Based Skeleton for Pairwise Biological Sequence Alignment," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, pp. 561-570, 2009.

[62] S. Lloyd and Q. O. Snell, "Hardware Accelerated Sequence Alignment with Traceback," *International Journal of Reconfigurable Computing*, pp. 10 2009.

[63] X. Meng and V. Chaudhary, "Boosting data throughput for sequence database similarity searches on FPGAs using an adaptive buffering scheme," *Journal of Parallel Computing*, vol. 35, pp. 1-11, 2009.

[64] M. N. Isa, K. Benkrid, T. Clayton, C. Ling, and A. T. Erdogan, "An FPGA-based parameterised and scalable optimal solutions for pairwise biological sequence analysis," presented at 2011 NASA/ESA Conference on Adaptive Hardware and Systems (AHS), 2011.

[65] J. Xianyang, L. Xinchun, X. Lin, Z. Peiheng, and S. Ninghui, "A Reconfigurable Accelerator for Smith Waterman Algorithm," *IEEE Transactions on Circuits and Systems II*, vol. 54, pp. 1077-1081, 2007.

[66] P. Zhang, G. Tan, and G. R. Gao, "Implementation of the Smith-Waterman Algorithm on A Reconfigurable Supercomputing Platform," Altera Corporation 2007.

[67] Y. Yamaguchi, H. Tsoi, and W. Luk, "FPGA-Based Smith-Waterman Algorithm: Analysis and Novel Design," in *Reconfigurable Computing: Architectures, Tools and Applications*, vol. 6578, *Lecture Notes in Computer Science*: Springer Berlin / Heidelberg, 2011, pp. 181-192.

[68] "Virtex-5 Family User Guide," Xilinx, Inc., San Jose, CA 2009.

[69] Y. Yoshiki, M. Yosuke, M. Tsutomu, and K. Akihiko, "High Speed Homology Search Using Run-Time Reconfiguration," in *Proceedings of the Reconfigurable Computing Is Going Mainstream, 12th International Conference on Field-Programmable Logic and Applications*: Springer-Verlag, 2002.

[70] X. Meng and V. Chaudhary, "A High-Performance Heterogeneous Computing Platform for Biological Sequence Analysis," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, pp. 1267-1280, 2010.

[71] "Virtex-5 Family Overview," Xilinx, Inc., San Jose, CA 2009.

[72] Y. Yamaguchi, et al., "FPGA-Based Smith-Waterman Algorithm: Analysis and Novel Design Reconfigurable Computing: Architectures, Tools and Applications," vol. 6578, *Lecture Notes in Computer Science*: Springer Berlin / Heidelberg, 2011, pp. 181-192.

[73] J. Xianyang, L. Xinchun, X. Lin, Z. Peiheng, and S. Ninghui, "A Reconfigurable Accelerator for Smith Waterman Algorithm," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 54, pp. 1077-1081, 2007.

[74] A. Krogh, M. Brown, I. S. Mian, K. Sjolander, and D. Haussler, "Hidden Markov models in computational biology. Applications to protein modeling," *J Mol Biol. 1994 Feb 4;235(5):1501-31.*, 1994.

[75] G. A. Churchill, "Stochastic models for heterogeneous DNA sequences," *Bull Math Biol. 1989;51(1):79-94.*, 1989.

[76] C. M. Stultz, J. V. White, and T. F. Smith, "Structural analysis based on state-space modeling," *Protein Sci. 1993 Mar;2(3):305-14.*, 1993.

[77] J. V. White, C. M. Stultz, and T. F. Smith, "Protein classification by stochastic modeling and optimal filtering of amino-acid sequences," *Mathematical biosciences*, vol. 119, pp. 35-75, 1994.

[78] S. R. Eddy, "Profile hidden Markov models," *Bioinformatics*, vol. 14, pp. 755-763, 1998.

[79] S.R.Eddy, "HMMER User's Guide," Washington University School of Medicine 2003.

[80] R. Hughey and A. Krogh, "Hidden Markov models for sequence analysis: extension and analysis of the basic method," *Computer applications in the biosciences : CABIOS*, vol. 12, pp. 95-107, 1996.

[81] P. Bucher, K. Karplus, N. Moeri, and K. Hofmann, "A flexible motif search technique based on generalized profiles," *Comput Chem. 1996 Mar;20(1):3-23.*, 1996.

[82] S. R. Eddy, "HMMER User's Guide Version 3.0," March 2010 2010.

[83] R. D. Finn, et al., "The Pfam protein families database," *Nucleic Acids Res. 2010 Jan;38(Database issue):D211-22. Epub 2009 Nov 17.*, 2010.

[84] D. Steven and Q. Patrice, "Hardware Acceleration of HMMER on FPGAs," *J. Signal Process. Syst.*, vol. 58, pp. 53-67, 2010.

[85] R. D. Finn, et al., "Pfam protein families database," *Nucleic Acids Research*, vol. 38, pp. 211–222., 2010.

[86] N. Abbas, S. Derrien, S. Rajopadhye, and P. Quinton, "Accelerating HMMER on FPGA using parallel prefixes and reductions," presented at The International Conference on Field-Programmable Technology (FPT) 2010.

[87] A. Di Bias, et al., "The UCSC Kestrel parallel processor," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, pp. 80-92, 2005.

[88] B. Schmidt, "Massively Parallel Sequence Analysis with Hidden Markov Models," presented at International Conference of Science Engineering Computing (IC-SEC), Singapore, 2002.

[89] E. GlÃ©met and J.-J. Codani, "LASSAP, a LArge Scale Sequence compArison Package," *Computer applications in the biosciences : CABIOS*, vol. 13, pp. 137-143, 1997.

[90] D. Lavenier and J.-L. Pacherie, "Parallel Processing for Scanning Genomic Data-Bases," *Proc. PARCO'97, Elseiver*, 1998.

[91] W. Zhu, Y. Niu, J. Lu, and G. R. Gao, "Implementing parallel hmm-pfam on the EARTH multithreaded architecture," presented at Proceedings of the 2003 IEEE Bioinformatics Conference, 2003. CSB 2003. , 2003.

[92] R. Hughey, "Parallel hardware for sequence comparison and alignment," *Comput Appl Biosci. 1996 Dec;12(6):473-9.*, 1996.

[93] P. M. Rahul, B. Jeremy, D. C. Roger, A. F. Mark, and H. Brandon, "Accelerator design for protein sequence HMM search," in *Proceedings of the 20th annual international conference on Supercomputing*. Cairns, Queensland, Australia: ACM, 2006.

[94] T. Oliver, et al., "Accelerating the Viterbi Algorithm for Profile Hidden Markov Models Using Reconfigurable Hardware ", vol. 3991, *Lecture Notes in Computer Science*: Springer Berlin / Heidelberg, 2006, pp. 522-529.

[95] A. C. Jacob, J. M. Lancaster, J. D. Buhler, and R. D. Chamberlain, "Preliminary results in accelerating profile HMM search on FPGAs," presented at IEEE International Symposium on Parallel and Distributed Processing (IPDPS) 2007.

[96] K. Benkrid, P. Velentzas, and S. Kasap, "A High Performance Reconfigurable Core for Motif Searching Using Profile HMM," presented at NASA/ESA Conference on Adaptive Hardware and Systems, 2008. AHS '08. , 2008.

[97] T. F. Oliver, B. Schmidt, Y. Jakop, and D. L. Maskell, "High Speed Biological Sequence Analysis With Hidden Markov Models on Reconfigurable Platforms," *IEEE Transactions on Information Technology in Biomedicine*, vol. 13, pp. 740-746, 2009.

[98] O. Tim, "High Performance Database Searching with HMMer on FPGAs," 2007.

[99] T. Oliver, L. Y. Yeow, and B. Schmidt, "Integrating FPGA acceleration into HMMer," *Journal of Parallel Computing*, vol. 34, pp. 681-691, 2008.

[100] S. Yanteng, et al., "HMMer acceleration using systolic array based reconfigurable architecture," in *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*. Monterey, California, USA: ACM, 2009.

[101] T. Takagi and T. Maruyama, "Accelerating HMMER search using FPGA," presented at International Conference on Field Programmable Logic and Applications, 2009. FPL 2009. , 2009.

[102] T. Oliver, L. Y. Yeow, and B. Schmidt, "Integrating FPGA acceleration into HMMer," *Parallel Computing*, vol. 34, pp. 681-691, 2008.

[103] M. Punta, et al., "The Pfam protein families database," *Nucleic Acids Research*, vol. 40, pp. D295, 2011.

[104] UniProtKB/Swiss-Prot, "UniProtKB/Swiss-Prot protein knowledgebase release 2012_11 statistics," SIB Swiss Institute of Bioinformatics 2011.

[105] S. Whelan, P. I. W. d. Bakker, E. Quevillon, N. Rodriguez, and N. Goldman, "PANDIT: an evolution-centric database of protein and associated nucleotide domains with inferred trees," *Nucleic Acids Research*, vol. 34, 2006.

[106] S. Yanteng, et al., "Accelerating HMMer on FPGAs using systolic array based architecture," presented at Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on, 2009.

[107] W. John Paul, et al., "MPI-HMMER-Boost: Distributed FPGA Acceleration," *J. VLSI Signal Process. Syst.*, vol. 48, pp. 223-238, 2007.

[108] Xilinx, "Virtex-5 Family Overview," Xilinx Inc DS110, 2009.

[109] Xilinx, "Achieving Higher System Performance with virtex5."

[110] C. M. Maxfield, *The Design Warrior's Guide to FPGAs*: Elsevier, 2004.

[111] W. R. Pearson and D. J. Lipman, "Improved Tools for Biological Sequence Comparison," presented at National Academy of Sciences of the United States of America, 1988.

[112] NCBI, "Basic BLAST " in *http://blast.ncbi.nlm.nih.gov [Last accessed: Jan 2013]*.

[113] L. Bleris, et al., "Improvement of BLASTp on the FPGA-Based High-Performance Computer RIVYERA," in *Bioinformatics Research and Applications*, vol. 7292, *Lecture Notes in Computer Science*: Springer Berlin Heidelberg, 2012, pp. 275-286.

[114] S. F. Altschul, et al., "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs," *Nucleic Acids Res. 1997 Sep 1;25(17):3389-402.*, 1997.

[115] E. L. Huzefa Rangwala, Roy Musselman, Kurt Pinnow, Brian Smith, Brian Wallenfelt, "Massively parallel BLAST for the Blue Gene/L," presented at High Availability and Performance Computing Workshop, 2005.

[116] L. Heshan, M. Xiaosong, P. Chandramohan, A. Geist, and N. Samatova, "Efficient Data Access for Parallel BLAST," presented at Proceedingsof the19th IEEE International Symposium on Parallel and Distributed Processing. , 2005.

[117] A. Jacob, J. Lancaster, J. Buhler, and R. D. Chamberlain, "FPGA-accelerated seed generation in Mercury BLASTP," presented at The 15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), 2007.

[118] L. Weiguo, B. Schmidt, and W. Muller-Wittig, "CUDA-BLASTP: Accelerating BLASTP on CUDA-Enabled Graphics Hardware," *IEEE/ACM TRANSACTIONS ON COMPUTATIONAL BIOLOGY AND BIOINFORMATICS*, vol. 8, pp. 1678-1684, 2011.

[119] P. Krishnamurthy, et al., "Biosequence similarity search on the Mercury system," presented at The 15th IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASSAP), 2004.

[120] M. Atabak and C. H. Martin, "Fast and accurate NCBI BLASTP: acceleration with multiphase FPGA-based prefiltering," in *Proceedings of the 24th ACM International Conference on Supercomputing*. Tsukuba, Ibaraki, Japan: ACM, 2010.

[121] L. Wienbrandt, S. Baumgart, J. Bissel, C. M. Y. Yeo, and M. Schimmler, "Using the reconfigurable massively parallel architecture COPACOBANA 5000 for applications in bioinformatics," presented at ICCS, 2010.

[122] S. Kasap, K. Benkrid, and L. Ying, "High performance FPGA-based core for BLAST sequence alignment with the two-hit method," presented at 8th IEEE International Conference on BioInformatics and BioEngineering, 2008. BIBE 2008. , 2008.

[123] E. Sotiriades and A. Dollas, "A General Reconfigurable Architecture for the BLAST Algorithm," *The Journal of VLSI Signal Processing*, vol. 48, pp. 189-208, 2007.

[124] J. C. Herbordt, J. Model, G. Yongfeng, B. Sukhwani, and T. VanCourt, "Single Pass, BLAST-Like, Approximate String Matching on FPGAs," presented at 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM) 2006.

[125] Xilinx, "Xilinx UG190 Virtex-5 FPGA User Guide," 2007.

[126] F. Zheng, X. Xu, Y. Yang, S. He, and Y. Zhang, "Accelerating Biological Sequence Alignment Algorithm on GPU with CUDA," presented at 2011 International Conference on Computational and Information Sciences (ICCIS), 2011.

[127] N. corporation, "GeForce 9600 GT," in *http://www.nvidia.co.uk/object/product_geforce_9600gt_uk.html [Last accessed : November 2012]*, 2012.

[128] L. Xiaoqiang, et al., "A Speculative HMMER Search Implementation on GPU," presented at The IEEE 26th International Symposium on Parallel and Distributed Processing & PhD Forum (IPDPSW) 2012.

[129] L. Xiaoqiang, et al., "A Speculative HMMER Search Implementation on GPU," presented at Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International, 2012.

[130] NCBI, "The NR Database," in *ftp://ftp.ncbi.nih.gov/blast/db/FASTA/nr.gz [Last accessed: Oct 2012]*.

[131] W. Liu, B. Schmidt, and W. M. ller-Wittig, "CUDA-BLASTP: Accelerating BLASTP on CUDA-Enabled Graphics Hardware," *IEEE/ACM TRANSACTIONS ON COMPUTATIONAL BIOLOGY AND BIOINFORMATICS*, vol. 8, 2011.

[132] F. Zheng, X. Xu, Y. Yang, S. He, and Y. Zhang, "Accelerating Biological Sequence Alignment Algorithm on GPU with CUDA," presented at Computational and Information Sciences (ICCIS), 2011 International Conference on, 2011.

[133] K. Benkrid, et al., "High Performance Biological Pairwise Sequence Alignment: FPGA versus GPU versus Cell BE versus GPP," *International Journal of Reconfigurable Computing*, vol. 2012, pp. 15, 2012.

[134] I. Berkeley Design Technology, "BDTI Certified™ Results for the AutoESL AutoPilot High-Level Synthesis Tool," 2010.

[135] Xilinx, "ZedBoard: Zynq-7000 EPP Concepts, Tools, and Techniques," 2012.

[136] Digilent, "ZedBoard Zynq™-7000 Development Board ", 2012.

[137] Xilinx, "Xilinx Introduces Zynq-7000 Family, Industry's First Extensible Processing Platform Dual ARM Cortex-A9 MPCore Processing System Tightly Integrated with Programmable Logic Extends Embedded System Architectures for Higher Performance and Scalability," 2011.

[138] H. Nabil, B. Ian, and J. Chris, "Scalable and partitionable asynchronous arbiter for micro-threaded chip multiprocessors," in *Proceedings of the 19th international conference on Architecture of Computing Systems*. Frankfurt, Germany: Springer-Verlag, 2006.

[139] Xilinx, "Xilinx Unveils the Vivado Design Suite for the Next Decade of 'All Programmable' Devices," 2012.

[140] M. Peter, et al., "Mapping of applications to MPSoCs," in *Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. Taipei, Taiwan: ACM, 2011.

[141] J. Teich, "From dynamic reconfiguration to self-reconfiguration: Invasive algorithms and architectures," presented at International Conference on Field-Programmable Technology (FPT), 2009.

[142] J. r. Teich, "Invasive Algorithms and Architectures Invasive Algorithmen und Architekturen," *it - Information Technology*, vol. 50, pp. 300-310, 2008.